Title

Higgs Manual
A Platform for Autonomy Research

Author

Carlos Hernández, Adolfo Hernando, Ricardo Sanz and Francisco Arjonilla

Reference R-2010-008

Release 0.1 Draft

Date December 5, 2011

Address

Autonomous Systems Laboratory
UPM - ETS Ingenieros Industriales
José Gutierrez Abascal 2
28006 Madrid
SPAIN

Higgs Manual

ASLab R-2010-008 v 0.1 Draft of December 5, 2011

Abstract

The target system selected for control is a Pioneer 2 ATX robot.

Keywords

Higgs, cognitive architecture, perception, action, integrated control, cognition, emotion, autonomy, autonomous systems

Acknowledgements

We acknowledge the support of the European Commission through grant *IST-027819-IP ICEA* of the IST-2004-2.4.8 Cognitive Systems domain and of the Spanish Ministry of Science and Innovation through grant *TIN2008-02250-E-1*.

Revisions

Release	Date	Content	Author
0.1	10/05/04	Initial release.	Sanz
0.2	28/09/11	User and developer manuals.	Arjonilla

Table of Contents

Chapter 1

Introduction

1.1 Document Purpose

The ASys search for universal cognitive architectures generalises insights gained in the study of animal brains, providing a model of cognition that effectively and homogeneously integrates autonomic, emotional and cogniitive aspects.

This document describes the Higgs robot, part of the ASys Robot Control Testbed (RCT) that has been selected as the target demonstration of the higher level cognitive dimensions into a physical system. The SOUL cognitive architecture intends the exploitation of croscutting design patterns to realise custom architectures targetted to specific uses. While SOUL leverages detailed knowledge about mammal brains as of integration of cognitive, emotional and autonomic aspects its main objective is to transcend the concrete brain implementations improving engineering capability for the construction of technical systems.

This implementation shall serve as vehicle for evaluating both the architecture and the componental technologies generalising the components extracted from biological models —e.g. rat brain parts like the amygdala, the basal ganglia or the hippocampus— that are addressed in other research activities.

1.2 Content

This report describes the Higgs Robot Control Testbed.

1.3 Intended Audience

This document is intended to provide a description of the concrete testbed for all stakeholders involved in development and evaluation of the SOUL architecture.

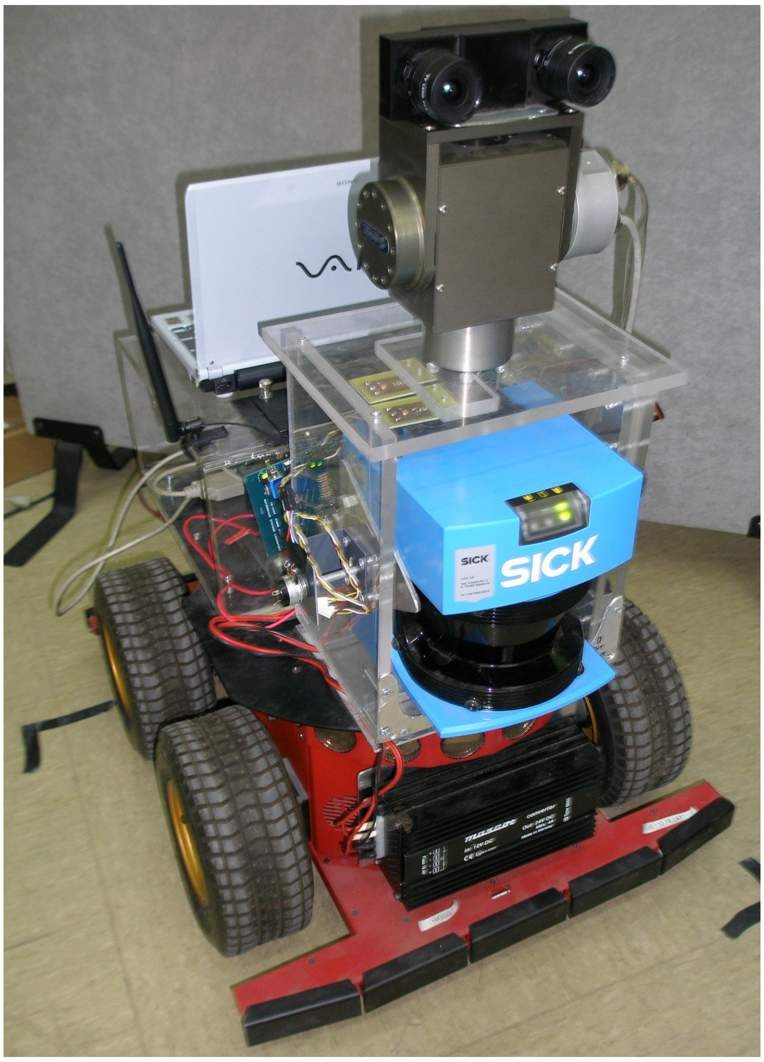


Figure 1.1: The Higgs robot is the central objective of development of the RCT, where the architecture for robust autonomy is demonstrated.

1.4 Mandatory References

The ASys project is the context for this work:

ASys Vision includes a description of the long-term objectives of the research.

OASys contains the ontology under development for ASys.

1.5 Structure of the report

This document purpose is to describe the implementation of a physical and software asset. It consists of several chapters roughly organised in four parts. Part 1 is dedicated to introductory materials that are useful to frame the main content of the report. Part 2 describes the hardware. Part 3 is the main content with the software and modelling description of the integrated architecture. Finally, Part 4 contains some appendices.

The content is as follows:

Chapter 1 provides an overall description of the document and the context for this work.

Chapter 2 contains the specification of the robot hardware.

Chapter 3 Describes the robot control software.

Chapter 4 contains an analysis of the mission.

1.6 Format

In the different chapters and section of this documents several systems or devices are described, so as to make it perfectly clear, the description of each one uses a tabular structure with the following formats:

This is mandatory text
`this depends of each device`
this is a comment.

1.7 Acknowledgements

This work has been partially supported by a European Commission grant to the project "Integrating Cognition, Emotion and Autonomy" (ICEA IST-027819) and by the "Conscious Cognitive Control" grant of the Ministry for Science and Innovation.

Chapter 2

The Higgs Robot Control Testbed

The Robot Control Testbed (RCT) is a collection of robot–based applications under development in the research group ASLab. The robots include commenrcial research platforms, simulated robotos and custom mobile robot implementations.

Higgs is a custom robot that is part of thr RTC developments. This is a mobile robotic system which consists of a base platform and different interconnected subsystems to cover a wide range of capabilities. The research aim is to provide the mobile robot with the necessary cognitive capabilities and an intelligent control system, as to perform complex tasks.

This manual includes information concerning its software and hardware description, specification, and use cases and additional information. The following sections summarise the key aspects of the testbed.

2.1 Higgs Structural Description

2.1.1 Base Platform

The base platform of Higgs is a mobile robot Pioneer 2–AT8 which has been designed by ActivMedia Robotics (Fig. 2.1). It is a robust platform which includes all the necessary elements to implement a control and navigating system, specially designed for outdoor applications. Additional systems and elements can be attached to this platform. The base platform is given the ASLab name Higgs, as a reference to the Higgs' bosson.

Higgs is a small size mobile robot, with a support structure made of aluminium. Its total weight is of 15 kg, being capable of carrying up to 40 kg. From a hardware viewpoint, the mobile robot consists of different elements:

Robot Panel: it is the superior platform of the robot, designed for a later assembly of new elements such as cameras or laser systems.

Robot Body: it is a box–shaped element made of aluminium. It contains the

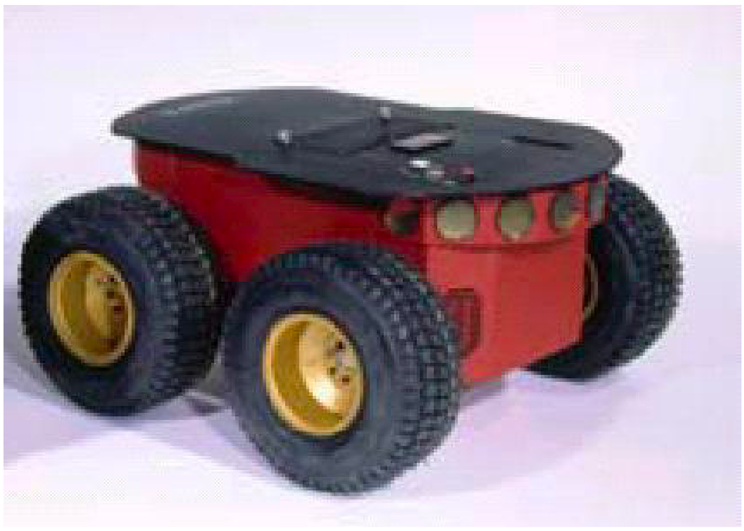


Figure 2.1: The Pioneer 2-AT8 platform used to implement the RCT Higgs robot.

batteries, the actuators, the electronic circuits and the rest of elements. It also allows to attach additional elements such as an onboard PC, a modem or additional sensors.

Control Panel: it is the access panel to the robot's microcontroller, placed in the robot panel. It consists of several control buttons, robot status leds (robot switch on, microcontroller status, battery charge) and a serial port RS–232 to be used as an input and output communication link with a external PC.

Sensors: the mobile robot is provided with two arrays of eight sensors each, which allow the detection and location of objects in the mobile robot environment. The arrays are placed at the front and at the rear part of the robot.

Actuators: the robots contains 4 Pittman motors GM9236E204. Each one includes an optical encoder to determine the robot's speed and position.

Microcontroller: it is a Hitachi H8S microcontroller consisting of different elements (memories, serial ports, inputs, outputs, 8 bit bus) which carries out different operations such as trigger and register the sensors' signal, control the actuators, and some other low–level operations.

Bumpers: they are additional elements attached to the platform, 5 at the front and 5 at the rear.

Power: there are three batteries 12 VDC 7 Ah–h, located at the rear part of the robot. They provide 252 W–h, which assure several hours of autonomy movement to the robot. Their status can be checked in the corresponding led of the control panel.

In addition to the hardware, the mobile robot has different software elements, either provided by the manufacturer or developed by ASLab team members.

AROS (ActivMedia Robotics Operating System): it is the operating system, consisting in server processes running on the Hitachi microcontroller in Higgs. It is a low–level software in charge of regulating the motors' speed, sonars' signal, encoders' signal and other low–level tasks. This software will also communicate the obtained information to other client software applications through the RS-232 serial interface.

ARIA: it is an applications–programming interface (API) based on C++ to control the robot. It acts as the client in the client–server topology. It allows to program high–level software applications, such as intelligent behaviour (obstacle avoidance, object recognition, wandering, exploration, etc). The robot control is based on direct commands, movement commands or abstract–level actions.

ASLab team software: there are a set of modules developed by the ASLab team members to extend or to add capabilities of Higgs. The modules developed are: communication [?], synthetic emotions [?], SOAR integration [?], voice [?], RT–CORBA control server [?], RT–CORBA robot status register, Java based CORBA client, CORBA based remote operation [?], and surface recognition [?].

2.1.2 Onboard Systems

On the base platform, different devices have been attached to expand the original range of functionalities of the mobile robot (Fig. 2.2).

- Onboard Computer: it is a computer attached to the base platform whose mission is to facilitate the communication with the microcontroller, the control of the robot, the execution of complex navigation operations, and additional communication operations. Taking into consideration several requirements and operational constraints defined in [?], a GENE–6330 was chosen. Related to software, the onboard computer uses Linux as operating system, as well as having a real–time software RTAI and RT–CORBA server.
- Laser: it is a laser scanner for mobile robotics applications, placed at the front part of the robot panel.
- Wrist: it is a 2 DOF high precision and torque actuator for orientation of the camera. It is attached to the laser.
- Camera: it is a stereo camera that will be used as the robot's vision system, attached to the wrist.
- Radio: it is a radio system for DGPS.

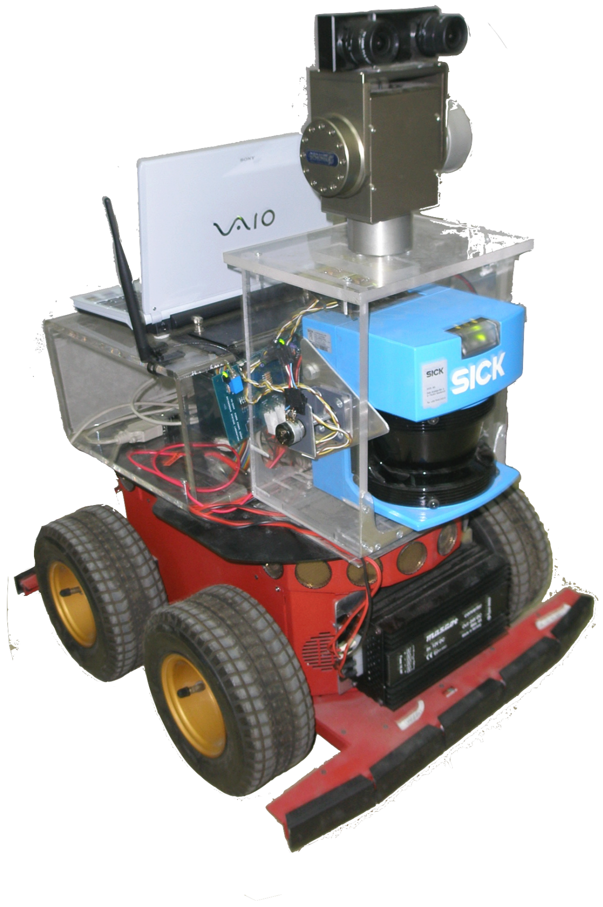


Figure 2.2: Pioneer 2-AT8 (Higgs) with onboard systems

- Laptop: it is a lightweight high functional laptop.
- GPS: it is a high performance GPS system.

2.1.3 Supporting Systems

Along with the base platform and the onboard systems, Higgs includes some additional supporting systems to complete its functioning.

1. Wireless Network: it is an additional subsystem included in the testbed to allow the communication with the wireless local network in the ASLab laboratory. It consists of a wireless card (placed in the onboard computer), an antenna necessary as the wireless card is placed inside the aluminium robot body (attached to the robot panel), and an access point (connected to the laboratory LAN).
2. Remote Control: it is a portable device to remote control the robot.
3. Server
4. User Terminal

2.2 Robot Functional Description

The functional description of the Higgs robot Testbed has been made through the specification of use cases and requirements (see Chapter 6). *FunctionalRequirement* has been defined in the ontology as a requirement that specifies an operation or behaviour a system must perform.

For the RCT, the functional requirements considered can be classified into primary and secondary ones. A primary functional requirement refers to a main capability to be fulfilled by the system. In the case of the RCT, two primary functional requirements have been considered: navigation, and survival.

A secondary functional requirement refers to an additional capability desired in the system. For the RCT, these secondary functional requirements are: to explore the environment, to avoid an obstacle while moving or exploring, reactive movement, the identification of an object in the environment, and search. Some additional functional requirements expand or detail the former ones, such as autonomous navigation as an extended case of the navigation one.

Chapter 3

Physical System Specification

The RCT testbed is a mobile robotic system intended as platform to test the technology developed in ICEA. The RCT platform is a pioneer mobile robot for research that is specially designed for outdoors applications. The aim of the RCT testbed is to purvey the system with the neccessary cognitive abilities so as to fulfil complex task. This abstract aim is grounded in the specification of a high level mission and the evaluation of the OM consciousness architecture as to enhenace mission level robustness.

In this document the overall situation of the RCT platform of the ICEA Project is presented. The purpose of the document is double:

- Report the current state of the platform.
- Provide a short description of the platform.

The specification section is presented in the form of tables of specifications to capture the main characteristics of the different devices that compose the RCT platform. For much more detailed information on the RCT platform please refer to Higg's Manual ?poner enlace?

3.1 Base Platform: Pioneer 2-AT8

3.2 Onboard Systems

3.2.1 Gene-6330

RCT Identifier:	RCT-Higgs
RCT Name:	Higgs
Description:	mobile robot for outdoors use
Functionality role:	Robot base
Model:	Pioneer 2AT-8
State:	fully operative
Serial number:	AT8CDBB1722
Purchase date:	200302
Physical Characteristics:	Alluminium body **Dimensions** 50x49x24 cm **Weight** 15 kg[1] **Payload** 40 kg
Physical links:	
Informational links:	**In** 5 analog **Out** 2 analog **In&Out** 3 RS-232 ports, 8 bits bus
Power:	3 batteries 12VDC 7A-h (252W-h)
Features: **Tech. doc.**:	**Microcontroller** Hitachi H8S **Actuators:** **Skid steering:** 4 pittman motors GM9236E204 **Sensors:** **Sonar:** 2 arrays (front&rear) each of 8 sensors, 20^o **Encoders:** 1 per wheel– 34,000 conts/rev **Bumpers:** 5 front and 5 rear

RCT Identifier:	RCT-Gene
RCT Name:	placa Gene
Description:	This is the onboard computer
Functionality role:	On board computer
Model:	GENE-6330
State:	Obsolete and retired
Serial number:	
Purchase date:	
Physical Characteristics:	**Dimensions** 146x101.6x26mm mm **Weight** 0.4 kg
Physical links:	attached inside Higgs
Informational links:	1 ethernet (RJ-45) — not used 1 configurable port RS-232/422/485 — not used 1 port RS-232 (need to put a conector) — not used Wifi (purveyed by Compact PCMCIA) — main external interface
Power:	typical 0.7W / Max 6.4W (+5V and +12V AT) from higgs batteries
Features:	**Microprocessor** Transmeta Crusoe TM5400, 600MHz **Memory** 64MB SDRAM + 256MB SDRAM **Flash Memory** CompactFlash I type 4GB **Wifi** purveyed by a Compaq WL110 PCMCIA + antenna
Tech. doc.:	

3.2.2 Laptop

RCT Identifier:	RCT-laptop
RCT Name:	vaio
Description:	a lightweith high functional laptop
Functionality role:	On board computer
Model:	Sony Vaio TX2HP
State:	Fully operative
Serial number:	
Purchase date:	
Physical Characteristics:	**Dimensions** **Weight**
Physical links:	The Vaio is attached to the back part of Higgs' top
Informational links:	**In&Out** 2 USB 2.0, wifi IEEE 802.11b/g, 1 ethernet, Bluetooth 2.0
Power:	from its internal battery (7.5 hour)
Features:	**Processor** Intel Pentium M 1.1GHz **Memory** 510 MB DDR2 SDRAM **OS** Windows XP and Fedora ?? **Hard drive** 80 GB -4200 rpm **Display** 11.1" LCD **DVD+RW**
Tech. doc.:	

3.2.3 Arduino

RCT Identifier:	RCT-arduino
RCT Name:	arduino
Description:	a simple board to conect some simple devices
Functionality role:	I/O board
Model:	
State:	Fully integrated
Serial number:	
Purchase date:	2007
Physical Characteristics:	**Dimensions** 175x55x50
Physical links:	Attached to the left side of the laser support
Informational links:	USB connected to laptop 3-wire connected to various sensors
Power:	6-12 VDC
Features:	**Sensors** 2 accelerometers, 1 compass, 2 current and voltage sensors, 1 servo, 1 potentiometer, 2 switches.
Tech. doc.:	

3.2.4 Laser

RCT Identifier:	RCT-Laser
RCT Name:	Laser
Description:	laser scanner for mobile robotics applications, SLAM, navigation, etc
Functionality role:	Scan device
Model:	Sick LMS-200
State:	Integrated
Serial number:	
Purchase date:	
Physical Characteristics:	**Dimensions** 155x156x265mm **Weight** 4.5kg^2
Physical links:	The laser support is screwed to the front part of Higgs' top. Includes a mechanism for tilting the laser by means of a servo. The RCT-Wrist is screwed to the top of the laser support
Informational links:	RS-232/**422** connected to vaio through USB/RS-232 converter
Power:	24VDC 40 W
Features:	**Scanning angle** 180^o **Restyp. meas. accuracy** 10mm/$\pm$35mm **Accesories** Metacrilate support $\pm 45^o$ tilt with a Futaba servo
Tech. doc.:	

3.2.5 Camera

RCT Identifier:	RCT-Camera
RCT Name:	Camara
Description:	Stereo camera for Higgs' vision system
Functionality role:	Camera
Model:	Videre STH-MDCS2-C
State:	available, already controlled independently but not integrated
Serial number:	
Purchase date:	
Physical Characteristics:	**Dimensions** 44x132x73 mm **Weight** 330 g
Physical links:	Screwed to RCT-Wrist
Informational links:	**In&Out** ieee 1394 (firewire)
Power:	< 1W - 12 VDC
Features:	**Accesories** RCT-Wrist
Tech. doc.:	

3.2.6 Wrist

RCT Identifier:	RCT-Wrist
RCT Name:	Wrist
Description:	2 DOF high precision and torque actuator for orientation (pan&tilt) of the RCT-camera
Functionality role:	Wrist
Model:	SCHUNK PW-070
State:	Fully integrated
Serial number:	B3844 & B3844
Purchase date:	
Physical Characteristics:	**Dimensions** 165x70x160 mm aprox **Weight** 1.8 kg
Physical links:	**axis2 appendix** attached to laser support **axis1 appendix** camera is attached here
Informational links:	**In&Out** RS-232
Power:	24 VDC, nomical power current 4 A per axis
Features:	**Sensors** 1 encoder per axis **Nominal torque** 12 Nm axis1, 2 Nm axis2 **Algle of rotation** 120^o axis1, 360^o axis2 **Resolution** 5″ axis, 6″ axis2
Tech. doc.:	

3.2.7 GPS

RCT Identifier:	RCT-GPS	
RCT Name:	GPS	
Description:	high performance GPS system	
Functionality role:	GPSd	
Model:	OEMV-2-RT2	
State:	ready to install	
Serial number:		
Purchase date:	2007	
Physical Characteristics:	**Dimensions**	60x100x13 mm
	Weight	56 g
Physical links:		
Informational links:	**1 USB**	connected to laptop
	1 RS-232/422	
	1 CAN bus	not used
Power:	1.6W typ. 12 VDC	
Features:	**Accuracy**	
	Accesories	RCT-radio
		Antenna
Tech. doc.:		

3.2.8 Power Board

RCT Identifier:	PowerBoard	
RCT Name:	Power Board	
Description:	A simple board for controlling power to other devices	
Functionality role:	Power board	
State:	Fully operative	
Serial number:		
Manufacturing date:	October 2009	
Physical Characteristics:	**Dimensions**	260x86x16 mm
Physical links:	Attached to the front of the Vaio support	
Informational links:	**Ribbon cable**	attached to arduino board
	Input 12V	connected to Inst. batteries
	12V-24V	to connect a 12V to 24V converter
	Input 24V	connected to 12V to 24V converter
	6x12V	Power for 12V devices
	3x24V	Power for Wrist, Laser and 1 free slot
	Power Jack	Battery charger
Power:	negligible, 12V to 24V	
Tech. doc.:		

3.2.9 Radio

RCT Identifier:	RCT-radio
RCT Name:	radio
Description:	a radio system for DGPS (GPS + difference with a known base = more precisssion)
Functionality role:	GPSd radio
Model:	Pacific Crest PDLRVR
State:	not installed
Serial number:	
Purchase date:	2007
Physical Characteristics:	**Dimensions** 21.0 cm L x 6.1 cm diameter
Physical links:	
Informational links:	
Power:	0.3W 9-16 VDC (from its own external battery)
Features:	**Weight** 0.34 kg **Accessories** Antenna Battery

3.3 Supporting Systems

3.3.1 PSP Remote Control

RCT Identifier:	RCT-PSP
RCT Name:	psp
Description:	a portable device with screen and controls to control remotely the RCT platform
Functionality role:	Remote controller
Model:	Sony PSP-1004
State:	available, not integrated
Serial number:	
Purchase date:	
Physical Characteristics:	**Dimensions** 170x23x74 mm **Weight** 280 gr
Physical links:	none (it's portable)
Informational links:	**In&Out** wifi (IEEE 802.11b), USB, infrared port
Power:	from its internal batteries
Features:	**Screen** LCD 4.3" (16:9) **Wifi Security** WEP, WPA (both AES & TKIP) **Video Compression** H.264/MPEG-4 AVC

3.3.2 Remote Server

3.3.3 Wireless Network

RCT Identifier:	aslab_wireless
RCT Name:	aslab_wireless
Description:	ASLab's wifi network at DISAM and its surroundings
Functionality role:	Wi-fi connection
Access point:	Linksys WAP54G
State:	fully operative
Serial number:	
Purchase date:	
Situation:	the access point is currently situated in the library
Encryption:	None
Features:	
Other:	config: usr= ; psswd=

Chapter 4

User Manual

4.1 Functionality description

The Robotic Control Testbed is a Mobile Robot built from a commercial robot base with added funcionality. The robot base consists of a four wheel vehicle with position and speed control embedded, integrated power and control electronics and 16 ultrasonic sensors that give readings on the distance to close obstacles. Additionally front and rear bumpers have been installed for safeguarding the robot while operating it in very constrained areas and for manual stopping.

The Laser range reader can measure distances from 8mm up to 80m, depending on configuration. Typically it will function with a resolution of 1mm and a range of 8mm to 8m. It is the main sensor along with the odometer readings in SLAM navigation. It can be rotated to make three dimensional maps with the servo. The mechanism operates in open loop mode, which gives a low precision angular rotation, given the mechanical hystheresys and the thermal and age deviations of set.

Next to the laser the I/O board, or data acquisition board, is a general purpose input output board that connects to a compass, a two axis accelerometer, the laser servo that controls its tilt movement, and two current and voltage sensors one for monitoring the consumption of the robot base and the other for the power board and related devices. These sensors and actuators are controlled through the I/O board module.

For outside applications there is a global absolute position sensor which consists of a GPS antenna, Radio receiver and battery for receiving differential corrections from a base station and a GPS receiver that processes the data and converts it to an easier format readable by the onboard computer.

Visual applications can make use of the binocular camera, which in combination with the wrist, that gives pan and tilt movement to the camera with respect to the robot base and is located on top of the laser, can direct the focus of the attention in almost any direction. The wrist is internally controlled in position, speed and acceleration, which gives precise and accurate motion control to the camera.

The onboard computer runs a full capable operating system and is the communications hub for controlling remotely all the devices on the robot. Software controlling the devices may be installed onboard or in a remote computer.

4.1.1 Visual identification of devices

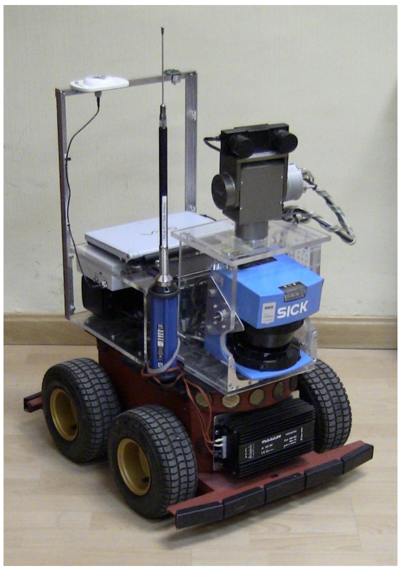


Figure 4.1: General view of Higgs.

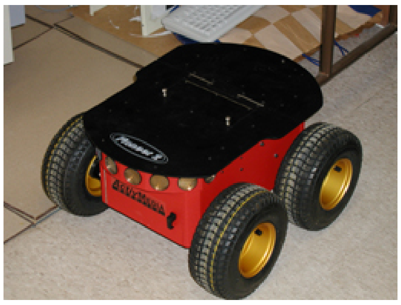


Figure 4.2: Robotic base Pioneer 2AT.

Figure 4.3: On board computer: VAIO laptop.

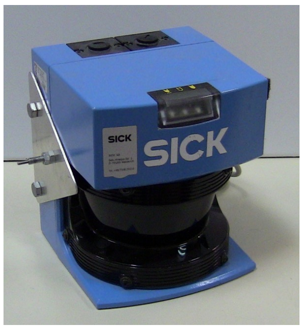


Figure 4.4: Laser sensor.

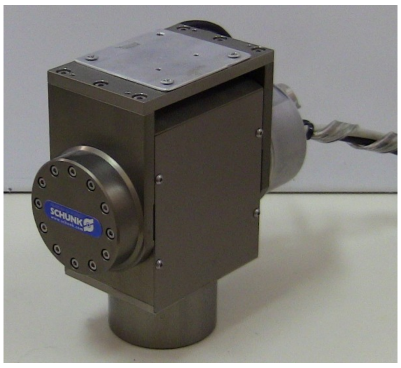


Figure 4.5: Wrist.

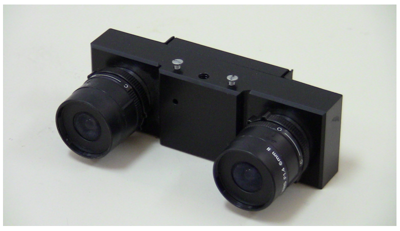


Figure 4.6: Camera.

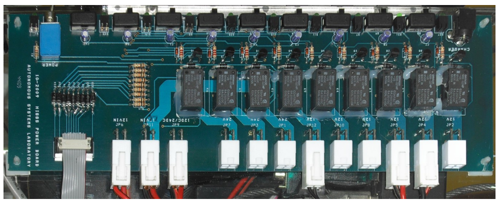


Figure 4.7: Power board.

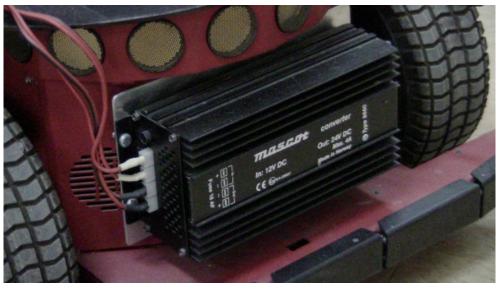


Figure 4.8: 12V to 24V converter.

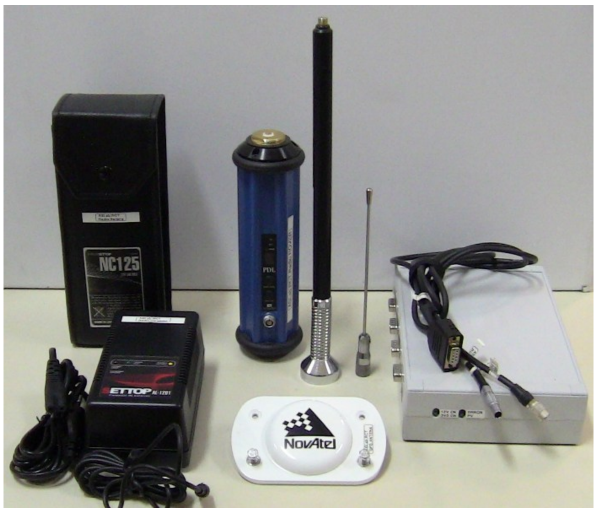


Figure 4.9: Differential GPS set. Rover station parts.

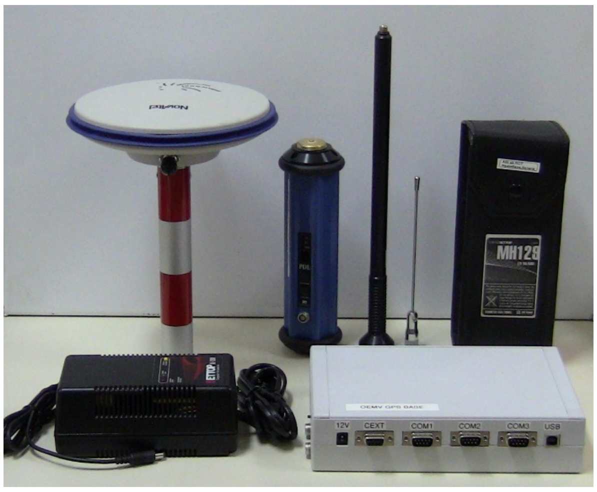


Figure 4.10: Differential GPS set. Base station parts.

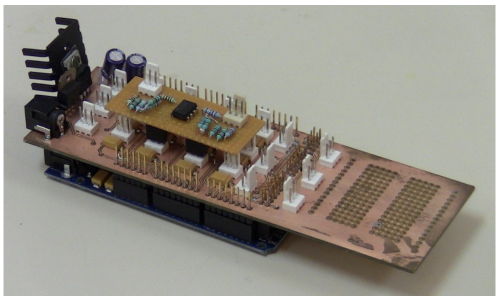


Figure 4.11: Input/Output board: Arduino.

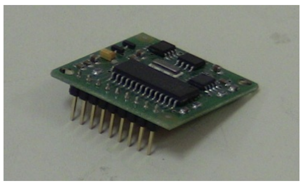


Figure 4.12: Compass.

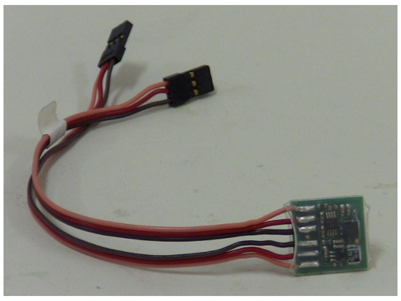


Figure 4.13: Accelerometer.

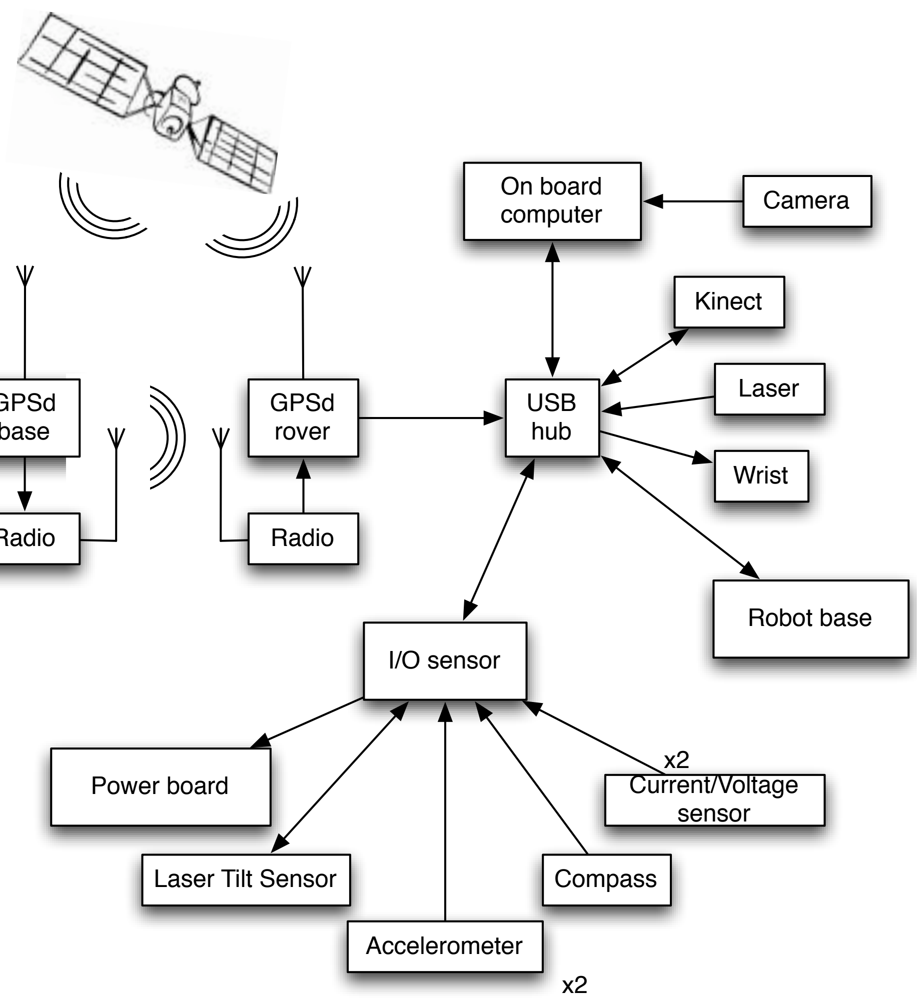


Figure 4.14: General data connection diagram

4.1.2 Data connection diagram

Arrows indicate the direction of valuable information flow.

4.2 Setting up the system

Preparing the robot for operation is easy. Power it on and all systems will start automatically. On next sections it will be discussed how to command it.

4.2.1 Powering the robot

There are three switches that must be turned on to enable the robot with full capabilities. The first one is at the back of the robot, next to the wheels. This switch powers the Pioneer2AT8 robot base and the power board. The power board has a general switch for all devices and one more for each device for manual disabling. The power board is placed between the two big methacrylate structures. Each switch has a label with the device that enables. The last device to power on is the VAIO laptop.

Note: The devices powered by the power board can be also disabled automatically by the Arduino. When the Arduino is not correctly powered on, the voltage levels at its pins are undefined, and may disable the power to some devices. Be sure to power the arduino if you are going to use any other device from the power board.

Only the laptop is mandatory to power on. The other switches may stay off if you are not going to use the device associated to that switch. However you will have to start at least one device to make it useful.

The Arduino does not disable any device by default. Both the manual switches and the Arduino may force the shutdown of any device, so to use a device, be sure that none of them disables it. The automatic switches are remotely controlled through the Arduino module.

4.3 Basic maintenance

There is little maintenance to do with the robot. The batteries are the most important matter to be aware, followed by the wheels.

4.3.1 Wheels

Once every two months or so, the wheels will loose pressure and they must be inflated evenly, this way the odometry will not loose precision. It can be noticed when the wheels have deinflated by looking to the tread pattern: Two separate bands of moist will indicate underinflation, one on the middle overinflation, and correct inflation when moist is evenly distributed. There is a manual pump with manometer in the cabinet.

4.3.2 Battery management

The robot has 3 battery packages. One of them is inside the Pioneer2AT8, the other one powers the radio receiver for the differential GPS and the last one is embedded in the laptop.

Robot base lead batteries

There are three lead battery packages inside the Pioneer2AT8 that powers the motors, the internal electronics and the external power board. To access them, lift the small black lever on the back side of the robot and turn it to the left. This will loosen the battery door. Open it to 135°. You may have to pull the door up to get the door over the bump sensors. Once it is open,

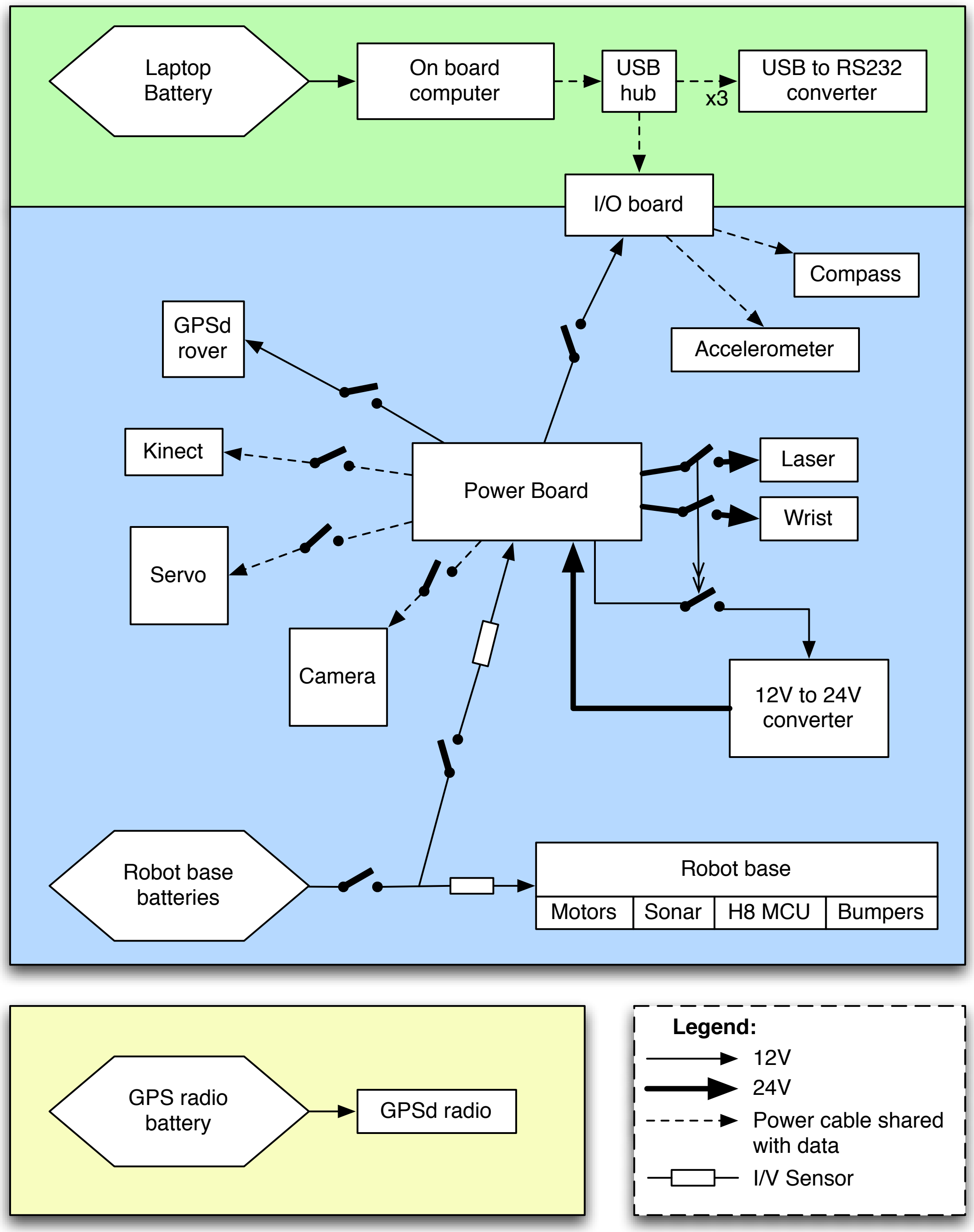


Figure 4.15: General power connection diagram

pull the batteries with the black sucker, or carefully with the hands do not get yourself pinched. When inserting the new batteries, remember that **the battery terminals go last**. Failure to introduce them correctly will cause short circuits and damage the electronics.

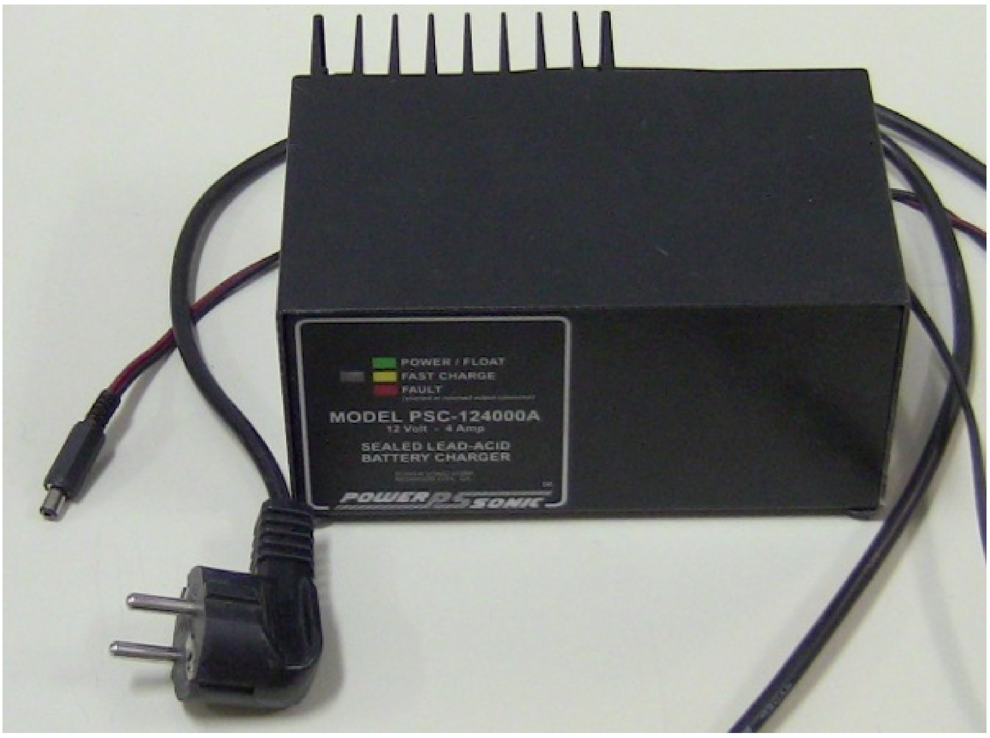


Figure 4.16: Robot base charger.

Recommended voltage when using the batteries is between 11.5V and 12.5V. If the batteries are too low the robot will indicate it with continuous short beeps. You can check the battery status through the battery LED in the Pioneer2AT8 panel: Green when fully charge through orange downto 11.5V, finally red. To keep the batteries in good conditions, do not discharge them completely. Doing so will decrease the charge capacity. It is better to fully charge them after each use.

To charge the batteries, connect the charger to the robot through the power jack connector to the left of the battery door. Its charger is the big black one, model PSC-124000A. You may leave the charger connected for as long as you want, but remember to **switch off the robot every night**, as electricity in the laboratory is turned off and the robot will fully discharge the batteries in this time, damaging them.

GPSD radio battery

There are two radio batteries, one for the GPS base station and the other for the GPS mobile station. Both batteries are interchangeable, and are 8x9x23cm in size with a black leather cover. Ni-MH batteries have memory effect, so you must discharge them completely before charging them again. There are two chargers. you may found them in the yellow suitcase in the “F room” or in a white long box next to the suitcase. Remember that the “F room” is the name for the ASLab cabinet in the computer rack room. To charge it, open the battery tab and connect the charger. Press the button for discharging it fully, then wait until the charger finishes the operation. Detailed instructions are printed on the charger.

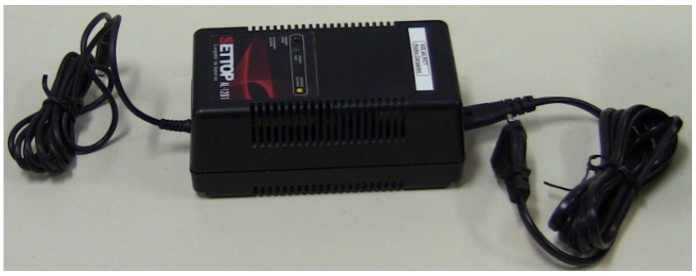


Figure 4.17: GPSD radio charger.

VAIO laptop battery

The VAIO laptop manages automatically its own battery. You may check its status with the command `acpi -b`. On linux RTAI, ACPI is not supported by the kernel and it will not work so this command is unavailable.

Figure 4.18: VAIO laptop charger.

4.3.3 Diagnostics

Power

There are several indicators of malfunction that can ve verified in case the robot does not work properly. In the first place, look for correct power in each device. This is how you can check it:

Pioneer2AT There is a red LED inside the Pioneer when it is powered, but can only be verified indirectly through reflections in the methacrylate structure that covers the top hole of the base. Alternatively, There is a dedicated red LED for this purpose in the Control Panel with the label *PWR*.

Laptop The VAIO laptop lights a green LED when it is on.

Laser When it is powered on, a green LED or two orange and red LEDs will be lighted in the front of the laser device. Green means powered and prepared, orange and red means powered and initializing.

Arduino When it is powered a little orange LED can be find behind the USB connector. A fast sanity check is to power the arduino and the servo. If everything is OK, the Laser will rotate to look upwards in a small angle.

Wrist There is no way to verify this device externally. However, it is quite robust and will normally work as supposed to. If everything is running ok, it will make a little calibration movement on startup.

Camera Depends on the camera used. The black stereoscopic one has a red LED that blinks when the driver is reading it and the Minoru3D lights up in white when the driver is reading it.

GPS There are two LEDs on the side. One of them indicates that power is OK and the other lights when enough satellites for position calculation are being tracked.

Power Board There is a green LED for the general switch and one for each device.

4.4 Operating the robot manually

4.4.1 Booting the onboard computer and choosing the OS

The onboard computer has several operating systems installed:

WindRiver OS `/dev/sda1` (1GB). This was once used as a testbench for the WindRiver OS. It is several years old and is not used any more.

Windows XP Professional `/dev/sda2` (25GB). The original Windows OS prepackaged with the laptop. The NovAtel GPSD utilities have been installed here for quick GPS diagnostics.

Ubuntu 10.04 Long Term Service `/dev/sda3` (14GB). This is the current working environment, with Real Time Application Interface (RTAI) kernel.

Fedora 13 `/dev/sda5 (29GB)`. Has the previous working environment, with CORBA modules and depending on the nameserver in the old onboard computer. Has a custom driver for supporting the FireWire camera.

Swap partition /dev/sda6 (100MB).

On bootup there is a selection of the operating system to start. It includes Ubuntu with and without real time kernel, a RAM memory test and the other three OSs. The default OS is Ubuntu with the RTAI kernel. Note that only the current working OS, that is, the Ubuntu with RTAI kernel, is documented here. The other operating systems are old but kept for backup and compatibility with old software.

Shutting down.

The correct procedure to shut down the onboard computer is to log in as root and request shutdown:

```
root@higgs2:/root# halt
```

Forcing instant shutdown using the power button is not recommended, however, normally there has been no problem with the OS afterwards.

4.4.2 Loggin in to the onboard computer through SSH

The Secure SHell server is run in the onboard computer by default and open to everyone that has a user account or the root password. The root password is the same as the root password for all the computers in the laboratory. Ask your tutor for a local account on higgs or the root password. Access can be obtained by opening a terminal and executing the command

```
your-pc:/$ ssh account_in_higgs@higgs2
```

and root access by changing `account_in_higgs` with `root`. If you want to execute programs that use a graphical interface, create a ssh tunnel for X with the option `-X`:

```
your-pc:/$ ssh -X account_in_higgs@higgs2
```

This way the program will execute in the onboard computer and display in your local screen remotely. In case you get this error while trying to connect:

```
 ssh: Could not resolve hostname higgs2:\
Name or service not known
```

then you have a problem with the name resolution on your computer. Copy these lines to `/etc/hosts` as root to solve it:

```
127.0.0.1 localhost
```

```
138.100.76.251  sagan.aslab.upm.es sagan

138.100.76.20   noe.aslab.upm.es noe
138.100.76.26   churchill.disam.etsii.upm.es churchill
138.100.76.250  quark.disam.etsii.upm.es quark
138.100.76.239  parker.disam.etsii.upm.es parker
138.100.76.252  aldiss.disam.etsii.upm.es aldiss
138.100.76.244  corea.disam.etsii.upm.es corea
138.100.76.243  mingus.disam.etsii.upm.es mingus
138.100.76.217  verne.disam.etsii.upm.es verne
138.100.76.241  gibson.disam.etsii.upm.es gibson
138.100.76.204  pohl.disam.etsii.upm.es pohl

138.100.76.15 ende.disam.etsii.upm.es ende

# 138.100.76.196  luna.disam.etsii.upm.es luna
138.100.76.196  arturo-desktop.disam.etsii.upm.es arturo-desktop

138.100.76.247 higgs.disam.etsii.upm.es higgs
138.100.76.246 higgs2.disam.etsii.upm.es higgs2
```

Now your computer knows how to translate the hostnames of the laboratory to IP's. Alternatively, if in a rush substitute `higgs2` with its IP, as in

```
your-pc:/$ ssh local_account@138.100.76.246
```

The machine with name `higgs` corresponds to the old on board computer which was embedded inside the robot base.

Once logged in to higgs2, applications can be run as in a standard linux distribution. Linux RTAI runs a normal kernel with standard functionality ontop of the realtime features. For using realtime, the programs must be loaded directly into the kernel as modules. All other programs run in soft real time.

On bootup, the onboard computer will connect automatically to the wireless accesspoint `aslab_wireless`. This accesspoint is in the same network as the other computers. The connection is configured using the ESSID of the wi-fi hotspot and the MAC address too.

4.5 Testing the modules

The CORBA servants access the NameService to publish their services[1]. The table 4.1 shows the "`id`"'s used by each module, being the "`kind`" parameter empty for all of them.

[1]If appropriate module is installed and running.

Description	IDL	Name Service ID
Camera	Camera.idl	CAMERA
I/O Board	Arduino.idl	Arduino
Robot base	Pioneer2AT.idl	PIONEER
Wrist	wrist.idl	wrist
Battery Model	BatteryModel.idl	BatteryModel
Current Monitor	BatteryModel.idl	CurrentAverage
Laser	Laser.idl	LASCOR
GPS	gps.idl	GPS

Table 4.1: Names of the CORBA objects registered in the NameServer.

4.5.1 Subversion.

The device modules have test programs that can be run either locally from the onboard computer or remotely without needing to log in. This section and the next one are a quick guide for reconfiguring and troubleshooting easy problems with the devices. Any problem not solved here requires further understanding of the robot software mechanisms and are described in the developer manual.

The first thing to do is download the source code. Supposing you already have installed the necessary programs and libraries, type

```
svn co svn+ssh://sagan/home/svn_repositories/Higgs
```

to check out the source. Again, replace `sagan` with 138.100.76.251 if your hosts.conf is not correctly configured and prepend it with your user name and an @ if your server user is not the same as the local user.

There is a second repository with older modules and code that have not yet been ported to the new CMake - subversion schema.

```
svn co svn+ssh://sagan/home/svn_root[/Higgs]
```

An even older CVS repository exists too.

4.5.2 Subversion directory hierarchy

Once finished, three directories will be available:

`trunk` The latest code available using the technology currently in development in the robot, which is ROS at the time of this writing.

`branches` Alternative code using other technologies, which currently is only CORBA.

`docs` The source code of this document.

This is the resumed directory tree inside `branches/CORBA`:

```
code
|-- LowLevelControl
|-- WorldModel
|-- batteries
|-- control_libraries
|-- devices
|    |-- arduino
|    |-- camera
|    |-- gps
|    |-- laser
|    |-- vaio_tools
|    `-- wrist
|-- idl
`-- lib
```

The directories to consider when programming new clients are:

`code/idl` Contains the IDL definition files of all the modules necesary for controlling the robot. You may open it for accessing the documentation for the interfaces and you will have to link to it for generating the stubs and skeletons.

`lib` C++ macro and CMake files (See section 4.6.2) for fast client development.

4.5.3 Checking the environment for starting test programs.

Configuration files

The next sections describe how to run the test programs contained in the CORBA branch for testing the devices.

Before running the test clients the configuration files must have been installed for proper operation. Clients only need one configuration file:

```
/etc/higgs/nameservice.ip
```

containing the address and port of the Naming Service that the servants are using for publishing themselves:

```
higgs2:9876
```

Configuring serial port links

On the server side there are a few more entries inside `/etc/higgs`, the most important one for configuration issues being `devices`. This directory contains soft links to the character devices that represent the USB to RS-232 converters in `/dev`. The servants open these files instead of the real devices so they can be reconfigured without having to recompile. Usually you will modify these links when swapping the serial cables of the devices or changing the arrangement of the USB to serial converters. Knowing which device file goes with which device is a matter of trial and error, starting the servants and testing wether they started or not. See 4.5.3 for more information on servant logging.

Starting and stopping the CORBA servants

The servants use upstart, the standard utility in Ubuntu for booting the system and running the daemons. You may start or stop the servants in case you need the correspondant devices, i.e. the laser and the robot base in ROS, or they are not automatically started on boot.

To start a servant use:

```
$ start higgs_device
```

and to stop it,

```
$ stop higgs_device
```

where `device` is one of laser, wrist, gps, arduino, pioneer or any other available CORBA servant.

There is one more daemon, the CORBA Naming Service, that opens the port 9876, is always running and does not interfere with the devices.

Logs and troubleshooting

Servants print their output to a log file in the on board computer placed at `/var/log/higgs`. You will have to consult these for debugging problems with the servants such as not starting, setting the device file and checking overall status.

4.5.4 I/O board

The next procedure is standard on all modules. Enter the directory
`$(HIGGS_ROOT)/branches/CORBA/code/devices/arduino/client`.
Be sure to have the complete source tree, at least the code subdirectory, as many methods rely on files situated back in the tree. Run:

```
cmake .; make
```

This will generate the CORBA stubs and headers and then compile the client code. The binary `arduino_client` will appear. Run it to read the parameters of the devices attached to the IO board and setreset some of the devices.

4.5.5 GPSd

You may want to test the full GPSd equipment with differential corrections or only the rover part. The differential readings may not work in the campus because of interferences in the environment that blocks radio communications between the base station and the rover.

Rover

the serial cables are correctly installed. The USB to RS232 converter should be attached to COM1. Turn on the GPS switch. Now to the software part: Go to `$(HIGGS_ROOT)/branches/CORBA/code/devices/gps/src` and run `cmake .; make`. Run `gps_client`. A command line menu will be printed form where you can check the satellites used in the solution, the current position and speed, the standard deviation for the position and the type of differential corrections used, if any.

Base station

The software part is the same as in the Rover part, with the standard deviation reducing to 0.02m or so if the differential correction is working. Preparing the base station:

Go to the ASLab closet and locate a yellow bag. Take out the black leather battery, the blue radio transmitter with the antenna, the white box housing the electronics and the cables. The GPS antenna is located on the roof with the coaxial cable hanging down the facade to the back of the room where Higgs lives. Open a window and take it inside. Beware of your workmates in winter! Connect it to the electronics box. Power the electronics box with 12V, for example using Higgs charger, and the radio transmitter to the battery pack using the appropriate cable. Finally connect the electronics box to the battery pack with the serial terminal attached to COM2. The battery pack is internally wired to connect the serial data and the power to the cable attached to the serial transmitter. Press power in the serial transmitter. After a few minutes, the Tx led should start blinking every second. This is the indication that the differential readings are being transmitted.

In the rover part, attach the three terminal cable to the blue radio receiver, the battery pack and the COM2 port in the rover GPS electronics box. Press the power button in the radio receiver and proceed as in the rover section.

4.5.6 Laser

Procedure is similar to the I/O board one. Go to `$(HIGGS_ROOT)/branches/CORBA/code/devices/laser/src` and `cmake .; make`. `laser_client` will be generated. Power the laser in the robot, wait for it to go green and some more seconds for the laser servant to start up, and run the client. A list with the distances of the latest reading will be printed to the console.

4.5.7 Wrist

Again, the procedure is similar, only in this case the device is not read but told the position to move to. Turn on the switch for the wrist and wait for a little calibration movement that indicates that it is ready. It will move to the reset position if not there before the calibration. Go to
`$(HIGGS_ROOT)/branches/CORBA/code/devices/wrist/src`.
Two clients will appear. The first of them, `wrist_client`, will move the two axis with incremental span around ten times then stop. The second one, `wrist_client_mouse_grab`, will let you control the two axis with the mouse. Once running, take the pointer to the center of the window to move the wrist to the starting position, then slowly move the pointer and the wrist will follow it. The range of movements is limited by the servant for secure operation whatever the input is. However, the batteries and converter may not be able to provide the required power if fast enough movements are requested.

4.5.8 Robot base

TODO

4.6 Developing a client

4.6.1 ASLab client utility functions for CORBA

The methods for initializing the CORBA infrastructure and getting the object references is quite complicated and always the same. The file `code/lib/CORBA_utils.h` contains C++ macros that ease the procedures for setting up a client, managing the errors, reading the object references from the nameserver and configuring itself for reaching that nameserver.

The typical client would be something like:

```
#include <iostream>
#include "implementationC.h"
#include "CosNamingC.h"
#include "Higgs/branches/CORBA/code/lib/CORBA_utils.h"

int main(int argc, char* argv[])
{
    CORBA_BEGIN_CLIENT(argc, argv);
    CORBA_GET_REFERENCE(module::implementation, impl, "IMPL");
    impl->do_things();

    CORBA_END_CLIENT;
    return 0;
}
```

With the correct route to `CORBA_utils.h`. The arguments for `CORBA_GET_REFERENCE()` are:

1. `module::implementation` The type of the object to fetch.
2. `impl` The identificator desired for the reference. It should not be defined nor declared, it is done inside the macro. Remember that references are used as pointers.
3. `"IMPL"` String with the name that the object is registered in the NameServer.

4.6.2 CMake

It is encouraged to use CMake as the tool for managing the compilation of the proyects. There is a sample CMakeLists.txt in `Higgs/branches/CORBA/code/lib` prepared for linking the CORBA libraries and generating the necessary dependencies for compiling. Substitute `_module` with the name of your module. The file `IDL_command.cmake` defines a macro for generating and managing the sources of the IDL interfaces. Include all interfaces that you need when calling the macro `MacroGenerateIDL`. Use the command

```
SET(IDL_DIR path/to/idl)
```

pointing to higgs' idl definition files to tell the macro where to find them.

Chapter 5

Developer Manual

The RCT testbed has been designed and developed with the tools and libraries that are currently, as the time of this writing, the state of the art. These tools are usually replaced by easier and more powerful ones with time, and Higgs can benefit from these. When such a change is performed, normally it will invalidate some of the documentation herein. Remember to write down the procedures and descriptions to match the new devices and/or software. When replacing a broken part, this chapter can be used as a guide for installing the new components. The tools and libraries in use are defined, as well as how they have been installed and configured, and a detailed description of the modules, both hardware and software.

5.1 Mobile base

5.1.1 Disassembling the robot

The main reason for disassembling the robot is to access the inside of the mobile base for repairs and updates. The remaining components can be accessed without major disassembling. The robot base has two black plates screwed to the red chassis with a joint between them. This allows to access different parts of the robot without fully disassembling it. The front part of the robot is where the motors are placed and the inner on board computer can be fitted. No computer is currently installed inside so it will rarely need servicing. See section 5.1.1 to access it. The back part houses all power and control electronics and the batteries and can be reached disassembling the back methacrylate structure with the alluminium gantry. This is achieved by unscrewing all six bolts fixing it to the robot base. Be sure to unplug all necessary cords when taking it out. The robot base will power up but will not work correctly if the multi-coloured ribbon cable is not properly connected to the black instrument panel of figure 5.1.

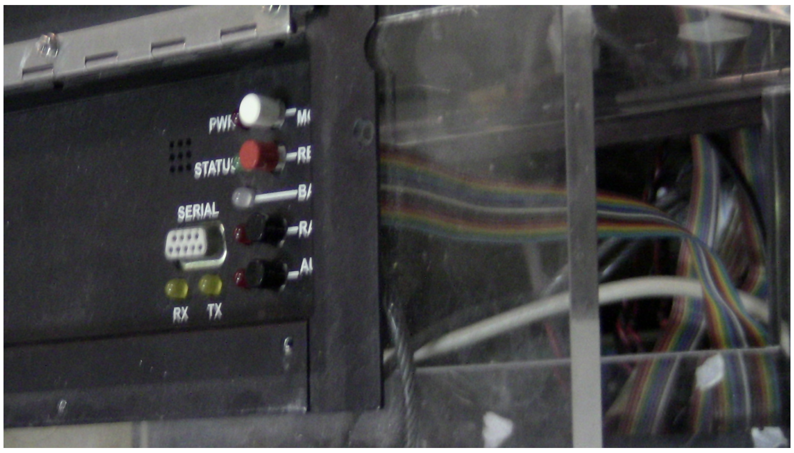


Figure 5.1: Robot base instrument panel.

Disassembling the laser

There are two ways to disassemble the laser. The first one removes the block composed of the laser sensor and the metacrylate structure that supports it. Get a number 5 allen wrench and unscrew the bolt placed under the laser, tiliting the latter upwards to reveal it. Then remove the two similar bolts on the other border of the alluminium plate that the first bolt was also holding. The methacrylate structure will be loose but do not take it off yet. Unplug the cords attached to the arduino board and the two data and power cords of the laser. Once all cables are released the structure can be lifted. Be careful not to damage the accelerometers when reassembling as its placement has been carefully crafted to fit between the laser structure and the arduino connection board.

The second way allows for easier but more tedious disassembling. Remove the radio receiver of the GPS turning it as it was a huge bolt. Then unscrew the four bolts holding the upper methacrylate plate of the laser structure. Be careful with the weight of the devices fixed to this plate. This will leave the two walls standing to the sides of the laser. Looking from a robot point of view, the right wall can slide outwards releasing the laser from its side constraints. Take it out after unplugging the power and serial data cords. If further disassembling is required proceed as in the previous paragraph.

Reaching the inside of the mobile robot.

Once the top part of the mobile base is free from devices and structures both front and back inside parts of the robot can be reached unscrewing the littlest black bolts on the black plates. Additionally you can also remove the sonar sensors for reaching deeper inside the robot. There are four little vertical bolts inside the structure that holds each sonar array. Remove the ribbon cable prior to disassembling it.

In the back part of the robot reside the electronics in two layers. To access the bottom boards the top ones must be removed unscrewing the four small bolts situated behind the back wheels at the side of the robot and removing the sonar array so it can be slipped out.

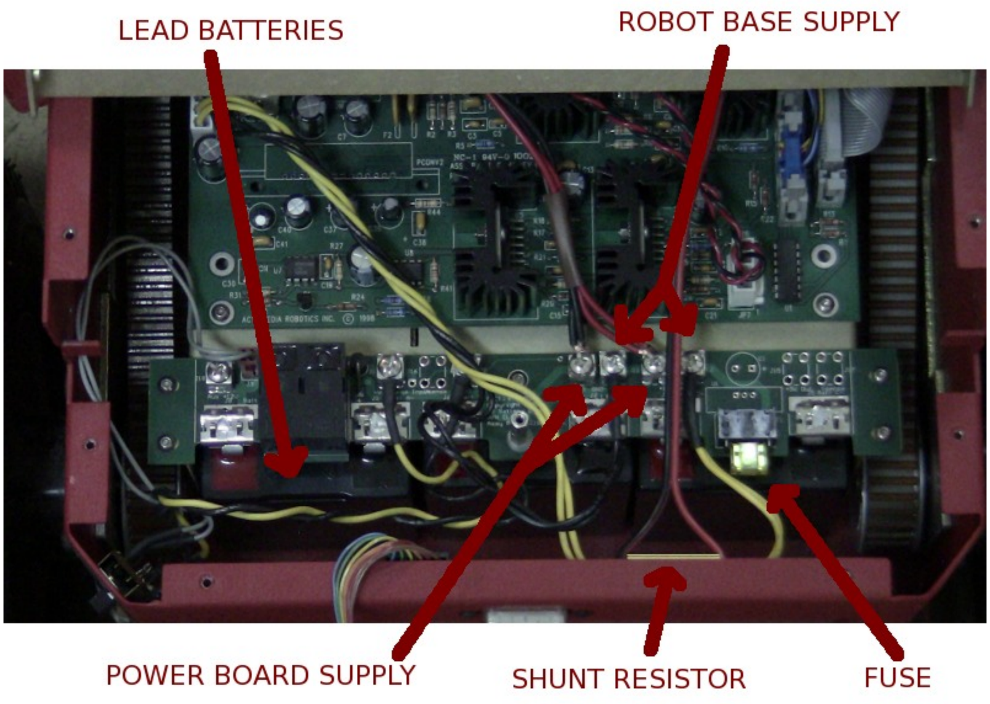


Figure 5.2: Robot base front part disassembled.

5.1.2 Firmware

It is possible to update the firmware of the Mobile base, uploading it to the nonvolatile memory of the Hitachi H8 microcontroller. Check the Pioneer 2 H8-Series Operations Manual. It is also possible to update the tick count of the odometry.

5.2 On-board computer

The onboard computer is the sony VAIO laptop running tUbuntu linux 10.04 with a RTAI kernel. The older onboard computer is a GENE board that was placed inside the mobile base with its own dc/dc regulator running WindRiver. See figure 5.3

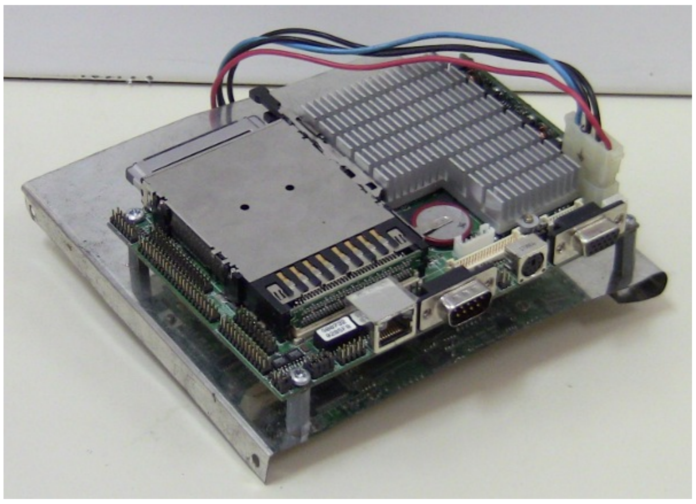


Figure 5.3: The old onboard computer GENE

5.2.1 Hubs and converters

The onboard computer has two USB ports and one FireWire port among others. The FireWire port is used with the binocular camera. The arduino, GPS, laser, wrist and robot base all use serial RS-232 communication which is achieved using USB to RS-232 converters. There is a hub behind the power board with 4 USB ports to where the converters are plugged, either directly (two of them), with a short USB cable (i.e. the GPS) or has the converter chip embedded inside the device (i.e. the data acquisiton board). There is a lack of one serial converter for having all five devices working at the same time. Be sure to reconfigure the serial ports at `/etc/higgs/devices` when changing the connected devices, see 4.5.3.

5.2.2 Real Time Operating System

This section describes the steps that have been made for building and installing the Ubuntu RTAI operating system.

1. Download and burn Ubuntu 10.04 LTS Desktop Edition, then install it like a standard Ubuntu installation. The Long Term Support surname guarantees that this version will be supported for 3 years.

2. Install these packages: `libncurses-dev build-essential kernel-package linux-source` They are needed for building the kernel and generating a debian package.

3. Download and uncompress the vanilla linux kernel 2.6.32.11. The version must much exactly so the RTAI patch can be applied smoothly. The linux kernel sources can be downloaded from `http://www.kernel.org/`.

4. Get the RTAI patch form `http://www.rtai.org/` As the time of this writing, the latest version was 3.8.1.

5. Apply the patch.

   ```
   cd /pathtolinuxkernel2.6.32.11/
   patch -p1 < /pathtortai/rtai-3.8.1/base/arch/x86/\
       patches/hal-linux-2.6.32.11-x86-2.6-0.3.patch
   ```

   Be sure to use the patch for the exact kernel vanilla version, in this case 2.6.32.11, or errors and warnings will arise. The `-p1` option removes the base directory form the path of the files inside the patch.

6. Configure the kernel. First, copy the ubuntu kernel config

   ```
    cp /lib/modules/`uname -r`/build/.config \
        /path_to_kernel-2.6.32.11
   ```

   and run `make menuconfig` inside the root directory of the downloaded kernel sources. Look for the following options and change them to the appropriate values: local version-append to -rtai.3.8.1-1; number of CPUs to 1; ACPI to no; all power management features to no, module versioning support to yes, interrupt pipeline to yes. The RTAI patch needs ACPI not to be supported by the kernel. As a consequence, no power management features will work and the kernel will not be able to run the HALT instruction on shutdown.

7. Compile with `make`. This can last many hours.

8. Generate the debian package.

   ```
   fakeroot
   make-kpkg --initrd kernel_image kernel_headers
   ```

9. Go to the upper directory, where the packages are created, and install them with `dpkg -i *.dev`.

Once the kernel is installed, reboot and start with the new kernel to check it con boot. Now it is time to finish the installation compiling the realtime kernel modules and utilities.

1. Go to the root directory of the RTAI source code and run `make menuconfig`. Select the number of target cpu's with the same number as the kernel and write the path to the kernel sources.

2. `make` and as root `make install`.

3. Configure the libraries as root.

   ```
   echo /usr/realtime/lib > /etc/ld.so.conf.d/rtai.conf
   ldconfig
   ```

4. Add `/usr/realtime/bin` to the path of all users' .profile files.

   ```
   echo "PATH=$PATH:/usr/realtime/bin" > ~/.profile
   ```

5. Finally, check that the realtime extensions are working correctly. Go to `/usr/realtime/testsuite/kern/latency` and run `./run`. If realtime is correctly installed, the last column should be all zeroes after 2-3 minutes.

6. Optionally, if space is low remove the packages and the source for the kernel and patch, or only the object files with `make clean`.

5.2.3 OS Tweaks

There are some modifications to be done to the Operating System to have Higgs' modules running flawlessly.

There must be a username called higgs and it must be a member of the dialout group. Note also that by default the iptables firewall is enabled on many distributions and it must be configured or disabled before CORBA clients can make calls to the servants.

Kernel modules

The linux kernel loads the serial modules for the converters at startup by default, but the order in which it does it is not the same on each bootup, so the serial device links must be checked on each bootup with this setup. The modules are the FTDI driver `ftdi_sio.ko` and the pl2303 driver `pl2303.ko`. This inconvenience has been avoided by moving the kernel file objects from their standard placement at `/lib/modules/` to `/etc/higgs/modules`. The kernel tries to load the modules but fails because it can not find them where it expects them to be. The modules are loaded with the init scripts in a determined order. The following commands as root can be used to set up this behaviour on new installations:

```
cd /etc/init.d
echo "#!/bin/sh" > load_serial_modules
chmod a+x load_serial_modules
echo "insmod /etc/higgs/modules/pl2303.ko" \
 >> load_serial_modules
echo "insmod /etc/higgs/modules/ftdi_sio.ko" \
 >> load_serial_modules
ln -s /etc/init.d/load_serial_modules \
 /etc/rc3.d/S60load_serial_modules
```

Replace `rc3.d` with the runlevel the OS starts with. You may instead prefer to modify the init script skeleton and give more functionality such as removal of the modules using the standard init V procedure such that loading and unloading the modules is done with `./load_serial_modules start/stop`.

Daemon scripts

The onboard portable computer is running many of the CORBA modules under Linux. The base operating system has been modified slightly to run in an unattended fashion.

Modifications made to the onboard computer:

- `/usr/bin/gnome_power_manager` has been renamed to `/usr/bin/gnome_power_manager.disabled`. This allows[1] for the computer to run while it is on batteries and the screen is down. Otherwise the Vaio goes into suspend mode and no process can execute. To read the battery life do
`cat /proc/acpi/battery/BAT1/state` or `acpi -b`

- `/etc/init.d` The servants are configured to be managed by upstart. The config files determine which modules to load, when to restart and where to redirect the output (logs). See section 5.3.2 for detailed description.

- *restart_servants* It is possible[2] to restart the servants by pressing the eject button on the Vaio, even with the lid down. This button will generate an ACPI event that will be processed by `acpid` using the configuration file `restart_servants.conf` which in turn will call `restart_servants.sh` that will force the termination of the servants, which will be restarted by upstart. The configuration and script files go respectively under `/etc/acpi/events` and `/etc/acpi/actions`.

- The output of the commands are redirected to `/var/log/higgs/$PROGRAM.log` Any problem associated to the execution of the modules may be diagnosed inspecting the logs in `/var/log/higgs/`, where all the output from the programs is registered. These files may grow big, so it has been created a logrotate config file for them (`higgslog`) placed in `/etc/logrotate.d/`.

- MODULES: Removed pl2303 and ftdi_sio kernel modules. They do not load with the same order on each bootup, so they are manually loaded by the inits scripts as described in 5.2.3. The binaries have been moved from the original location to `/etc/higgs/modules`.

5.3 Common libraries and module considerations

The RCT testbed may be controlled remotely by means of procedure calls to CORBA objects. All of Higgs devices including the base platform have a

[1]This is not valid in Ubuntu RTAI as the ACPI subsystem is not functional. It is documented here in case other OS is used.
[2]This is not available when running the RTAI kernel.

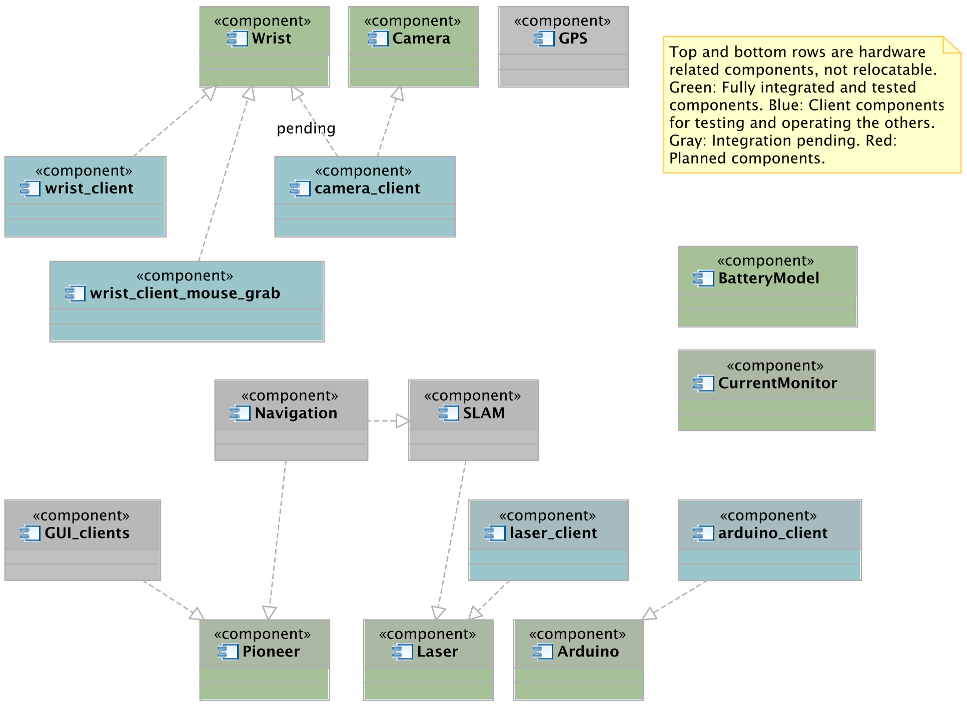

«component» Wrist
«component» Camera
«component» GPS
Top and bottom rows are hardware related components, not relocatable. Green: Fully integrated and tested components. Blue: Client components for testing and operating the others. Gray: Integration pending. Red: Planned components.
pending
«component» wrist_client
«component» camera_client
«component» wrist_client_mouse_grab
«component» BatteryModel
«component» CurrentMonitor
«component» Navigation
«component» SLAM
«component» GUI_clients
«component» laser_client
«component» arduino_client
«component» Pioneer
«component» Laser
«component» Arduino

Figure 5.4: General view of the CORBA modules.

CORBA interface definition that must be used in order to remotely or automatically operate the robot. In this section these interfaces are described in detail.

There are additional CORBA objects running in the on board computers that whilst not having direct control on Higgs' physical devices, they do use the devices and provide useful, well-known and proved services.

The Java language requires that all CORBA interfaces are defined inside a module, so all Higgs interfaces are defined inside the module `higgs`.

5.3.1 ASLab servant utility functions for CORBA

The same file `code/lib/CORBA_utils.h` for easy creation of clients has also macros for servants. A typical servant executable would be something like:

```
#include "implementation.h"
#include "CosNamingC.h"
#include "../../lib/CORBA_utils.h"

int main(int argc, char* argv[]) {
    CORBA_BEGIN_SERVER(argc, argv);
    implementation_t impl();
    higgs::implementation_t_var implvar = impl._this();

    CORBA_REGISTER_REFERENCE(implvar, "IMPL");
    CORBA_END_SERVER;
    return 0;
}
```

It is possible to use also `CORBA_GET_REFERENCE` in case other objects are needed.

5.3.2 Module installation and configuration files

The CORBA macros for the servants make use of the file

```
/etc/higgs/listen_endpoint.ip
```

to configure the ip address where they should listen to, additionally to `nameservice.ip`. Typically will contain the IPv4 address of the computer it is running in, in this case,

```
138.100.76.246
```

Other config files are the device links at

```
/etc/higgs/devices
```

each module has this directory and the device file it uses hard coded.

Finally, they need an upstart config file. The upstart config file for the laser follows as example:

```
description "Upstart config file for the arduino servant"
author "Francisco J. Arjonilla Garcia"
start on started Naming_Service
respawn
script
sleep 5
date >> /var/log/higgs/laser.log
su -l -c /usr/local/bin/laser_server higgs\
           >> /var/log/higgs/laser.log 2>&1
end script
```

Other modules have different upstart config files. See each module source tree. The Naming Service should start automatically too after installing it. If not, an upstart config file must also be created for it. In either case, the endpoint must be specified in the parameters, i.e. manual execution:

```
./Naming_Service -ORBEndPoint\
    iiop://higgs2.disam.etsii.upm.es:9876
```

The JAVA servant does not use the automatic NameService resolution mechanism provided by the C++ macros in `CORBA_utils.h` and needs to have it specified as arguments when running it. Also, the next two lines must exist in the `.profile` file in the home directory of the user running the servant:

```
export PATH=/opt/jdk1.6.0_26/bin:${PATH}
export LD_LIBRARY_PATH=/usr/local/lib/
```

Replace paths apropriately.

5.4 I/O board

The I/O board manages the sensors and actuators that do not have a specific interface for connecting them to a computer. It has been implemented with an Arduino Mega board. The devices controlled by the I/O board are:

- Compass
- Accelerometers
- Battery sensors
- Laser pitch
- Power board

These devices are connected to the Arduino Mega commercial board through a custom made extension board. The connector layout is shown in Figure 5.5, and the connectors correspondence to the devices in Table 5.1. The ribbon cable connector is between the digital inputs. The position is marked on the board and pin 0 (ground) is next to Pin22.

5.4.1 Tilt mechanism for the laser

The laser sensor is housed inside a methacrylate structure with a joint that enables pitch movements on the laser. They are actuated by a servo placed just behind the laser, and by two end of stroke switches that limit the movements to safe values. To the left side of the laser, coaxial with the axis of rotation, there is a potentiometer that closes the loop for precise control of the rotation angle. The servo, switches and potentiometers are all controlled by the I/O board and has a PID programmed within its firmware. The details can be found in the PFC from Marcos Salom.

Currently the PID feature has been disabled and the servo operates in an open loop fashion, using the end of stroke switches as a reference for rotation limits during initialization. The factors that motivated this decision are:

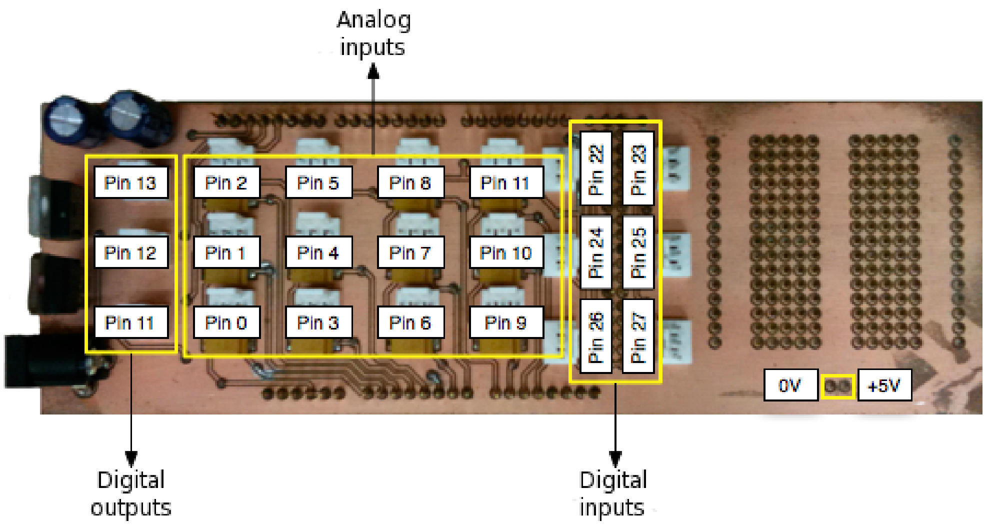


Figure 5.5: Connection diagram for compass board

Number	Device
Pin0	Not used
Pin1	Not used
Pin2	Not used
Pin3	Instrumentation current
Pin4	Instrumentation voltage
Pin5	Laser pitch potentiometer
Pin6	Motor current
Pin7	Motor voltage
Pin8	Accelerometer, X axis
Pin9	Not used
Pin10	Not used
Pin11(Analog)	Accelerometer, Y axis
Pin11(Digital)	Not used
Pin12	Not used
Pin13	Laser pitch servo
Pin22	Min pitch switch
Pin23	Max pitch switch
Pin24	Not used
Pin25	Compass
Pin26	Not used
Pin27	Not used
Ribbon	Power board

Table 5.1: Connector correspondence of the Arduino extension board with the devices.

1. The potentiometer is prone to errors due to mechanical hystheresis, worn out and temperature effects.
2. There is a lot of functionlity that can be more useful than controlling the pitch of the laser. The effort has been so displaced elsewhere.
3. A better pitch angle sensor, such as an optic encoder, would justify buying a whole controlled actuator with integrated sensing and electronics.
4. The kinect sensor can partially substitute the laser.

5.4.2 Power board

The I/O board and the power board have been connected with a ribbon cable that plugs to the doble row of pins of the extension board in the side of the I/O board. The ground connection has been stripped as the correspondent pin is not connected to ground in these pins. Common ground with the rest of the robot is achieved through the power connections that come from the batteries (Figure 5.2).

5.4.3 Compass

The compass is a solid state magnetism sensor placed next to the GPS antenna on the aluminum bridge. It was bought at www.superrobotica.com product reference CMPS03 S320160. The connections are as follows, from bottom to top as shown in figure 5.6.

1. VDD. To Arduino supply.
2. SCL. To 5V with 47Ω resistor.
3. SDA. To 5V with 47Ω resistor.
4. PWM. To Arduino digital input.
5. NC.
6. Calibrate. To button in compass connector.
7. 50Hz-60Hz. To GND.
8. NC.
9. GND. To Arduino supply.

All connections including the calibration button are done inside the IDT connector and protected by thermoadhesive. The button takes the calibration input to ground when pressed.

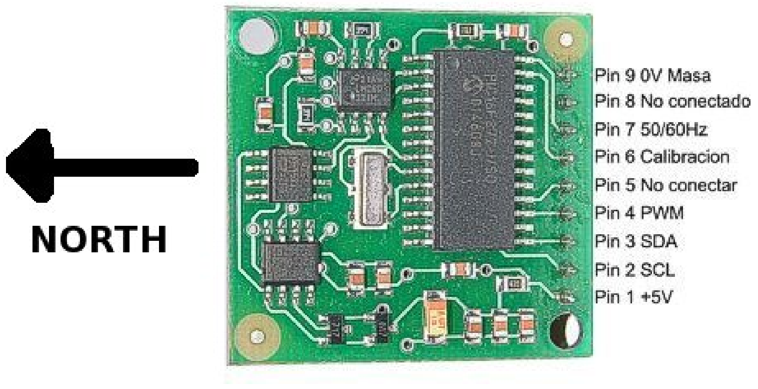


Figure 5.6: Connection diagram for compass board

Calibration instructions

As noted in the manual of the compass, to calibrate it you have to press the button with the board heading perfectly well once in each direction: North, West, South and East, no order required. It is already calibrated, so there is no need to do it again.

The compass gives an output pulse of 1ms to 37 ms VCC plus a fixed 65ms GND. 1ms corresponds to $0°$ and 37ms to $359°$.

The aluminum bridge does not interfere on the compass readings, but if the robot is moving near steel structures they may be perturbed.

5.4.4 Accelerometers

There are two accelerometers encased on the same electronic board attached horizontally with Velcro to the top cover of the robot base next to the I/O board. They give a standard analog 0V to 5V signal and are connected directly to the I/O board.

5.4.5 Intensity/Voltage sensor

There are two I/V sensors installed on board plus the battery status indicator in Vaio accessible through the command line. The two sensors can detect current and voltage of the Pioneer2AT batteries and the instrumentation batteries. One operational amplifier per battery has been used as a differential amplifier, as shown in Figure 5.7. This design allows to detect small variations in voltage of higher voltage than those of the working conditions of the operational amplifier, specifically lower than $3.5V$, with the LM2904P operational amplifier powered at $5V$. The voltage sensor is a scale down of the battery voltage, which is then divided by the scale factor by software to recover the true value. The gain is that of the differential amplifier. To read the true current value, both the gain and the sensing resistor value must be considered. See Table 5.2 for the numerical values. The printed circuit board is designed to fit into connectors 3,4,6 and 7 of the Arduino extension board as an add-on module. The sensing resistors are placed one inside the Pioneer2AT glued to the chassis under the back sonars (figure 5.2) and the other one is in an aerial connection on the battery cable immediately before the power board.

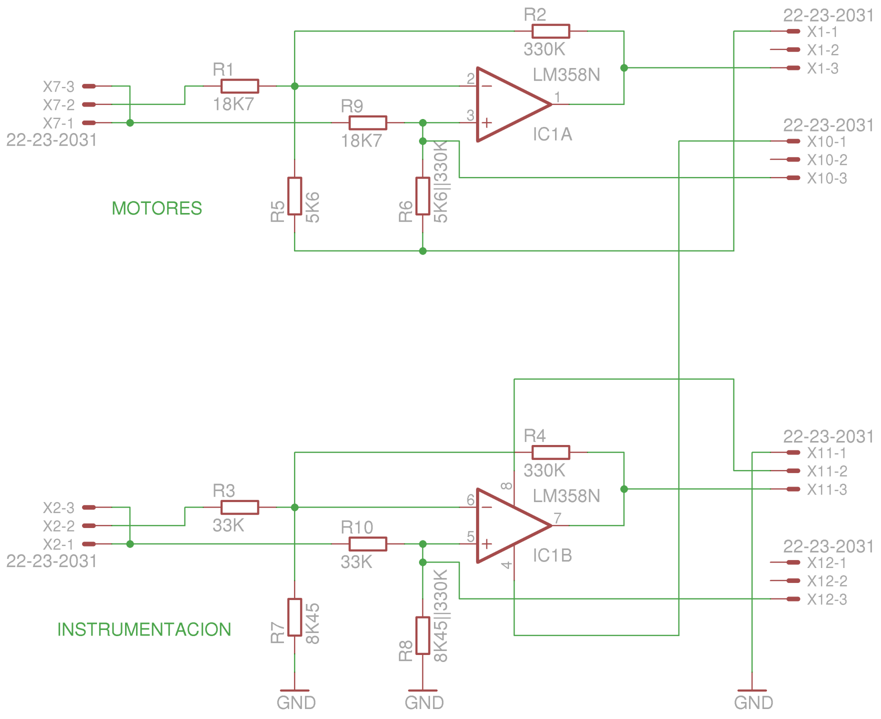


Figure 5.7: Battery sensor schematic

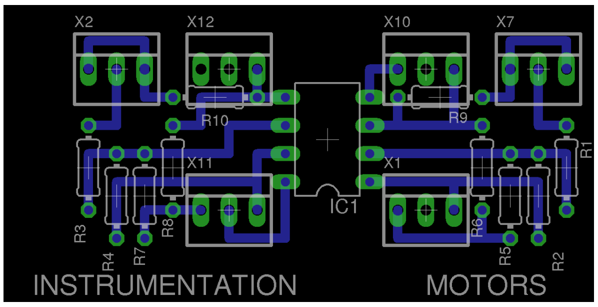


Figure 5.8: Battery sensor board layout

Supposing ideal components, the output voltage given by the current sensor is given by the formulae

$$U_0 = U^{+}\frac{R_6 R_2}{R_9 + R_6}\left(\frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_5}\right) - U^{-}\frac{R_2}{R_1} \tag{5.1}$$

The condition that must be met for both references to be scaled down by the

Parameter	Instrumentation	Motors
R1	$33k\Omega$	$18.7k\Omega$
R2	$330k\Omega$	$330k\Omega$
R5	$8.45k\Omega$	$5.6k\Omega$
R6	$8.45k\Omega//330k\Omega$	$5.6k\Omega//330k\Omega$
R9	$33k\Omega$	$18.7k\Omega$
Max sensing current	6A	17A
Max differential input	0.3V	0.17V
Gain	10	17.647
Voltage scale factor	0.1998	0.2275
Sensing resistor	0.05Ω	0.01Ω

Table 5.2: Characteristics of the differential amplifiers for the battery sensors.

same coefficient is that the relation of the resistor values between the positive input and the negative input is

$$\frac{R_9}{R_6} = \frac{R_1}{R_2 \parallel R_5} \tag{5.2}$$

This way,

$$U_0 = \frac{R_2}{R_1}(U^+ - U^-)$$

From this result we can observe that there is great sensibility in the operation of the differential amplifiers. The resistors from the negative input and the positive input must be of the same value, of the same brand and of the same batch to give enough precision: $R_9 = R1$ and R_6 physically being two resistors in parallel: $R_6 = R_2 \parallel R_5$.

The current sensors will not work correctly while the batteries are charging.

5.4.6 Servant and firmware

The CORBA module has been written in the JAVA language whilst the test client in C++. When the TurnOn and TurnOff methods are called, the appropriate device is turned on/off and simultaneously, for the gps, laser, wrist and camera, the servant that controls the device is killed from the arduino servant This only works if the device servant is installed and running on the same machine as the arduino servant, and will solve some issues on the protocol management that some of the servants have with their device. The servant will then be restarted by the vaio tools utility scripts.

Both the embedded program inside the I/O board and the Java servant communicate through a USB data connection that gets converted to serial RS-232 by a FTDI chip in the Arduino board. The I/O board starts the communication protocol by sending all the parameters and sensor readings to the servant, and then the servant optionally answers with the order.

The figures 5.9, 5.10, 5.11 and 5.12 show the UML model of the JAVA sources of the Arduino CORBA servant.

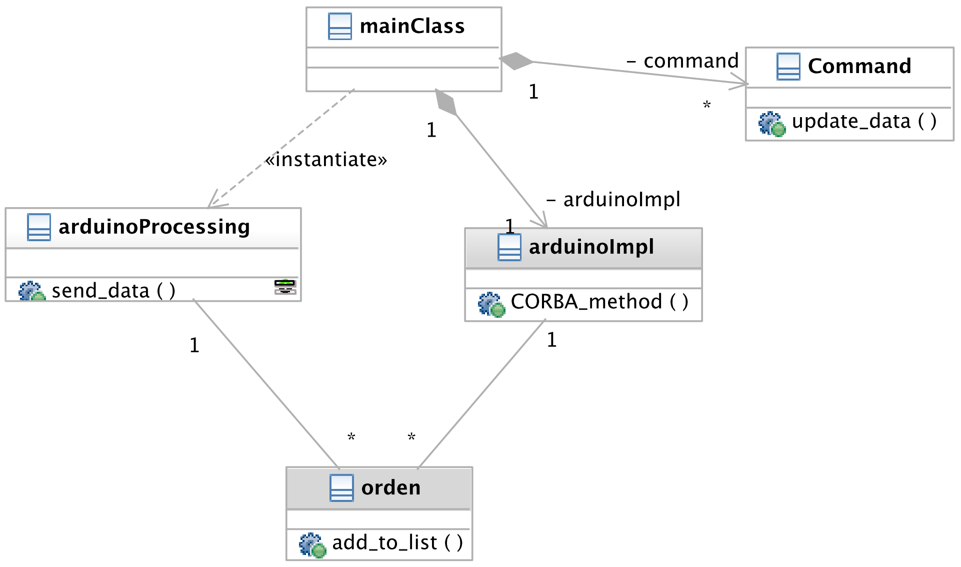


Figure 5.9: CORBA interface and classes used in the JAVA implementation.

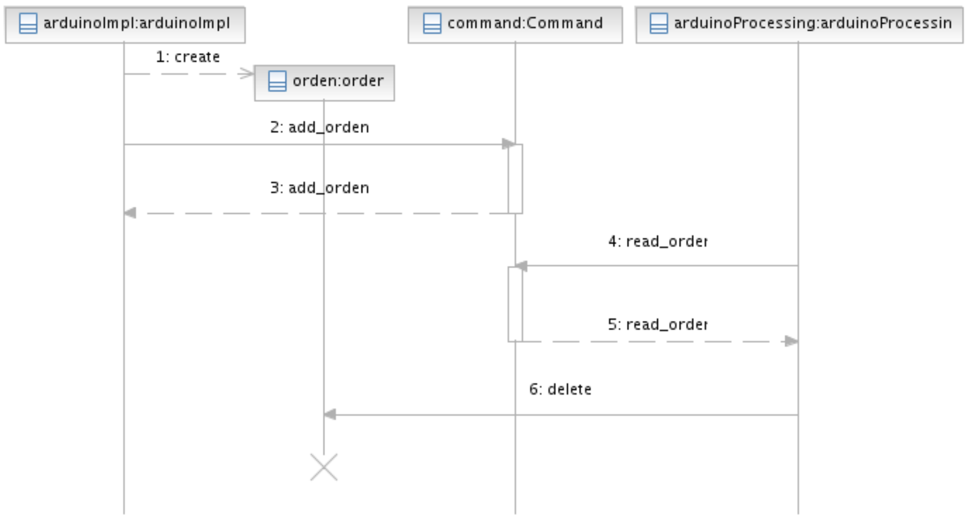


Figure 5.10: Interaction diagram for the Arduino CORBA module.

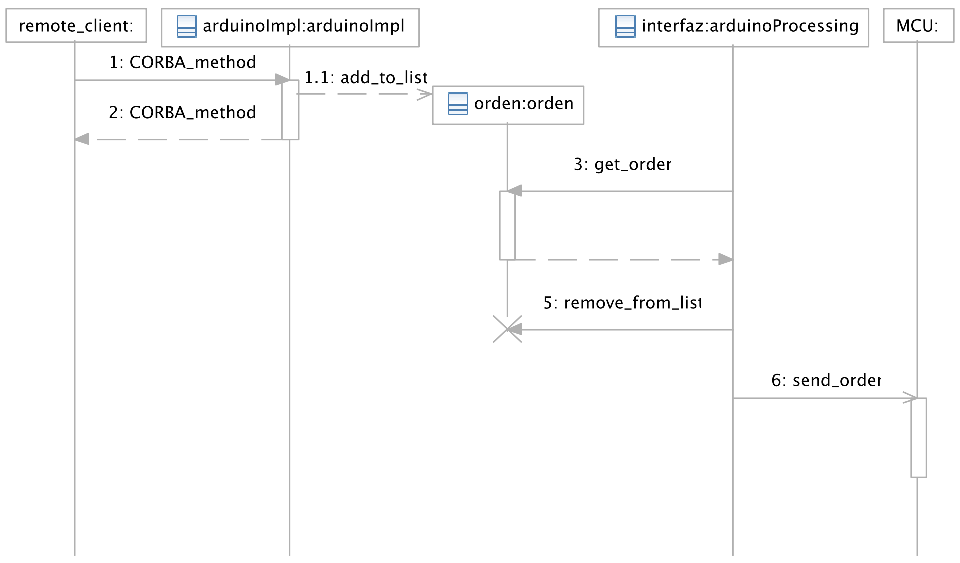


Figure 5.11: Sequence diagram for sending data to the i/o board.

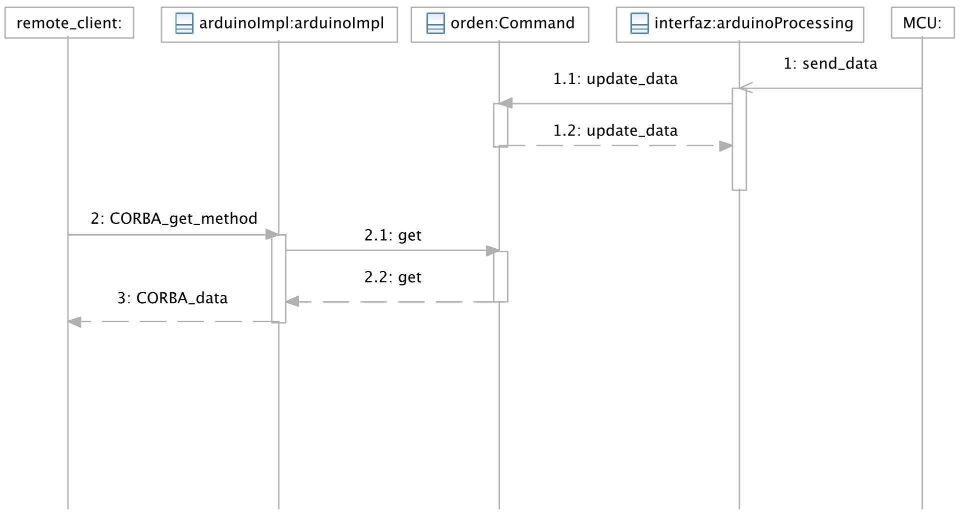


Figure 5.12: Sequence diagram for receiving data from the i/o board.

Modifying and uploading the embedded C code

The embedded program inside the Arduino has been developed using the official IDE based on the *processing* IDE and the libraries associated to it.

There are a few issues for remembering when installing this module on a fresh linux installation. The serial library RXTXcomm is included in the subversion directory
`svn+ssh://sagan/home/svnroot/Higgs/code/devices/arduino/lib`
The file librxtxSerial.so should be copied to
`/usr/$JAVA_BASE/jre/lib/i386` and the three jar files `core.jar RXTXcomm.jar` and `serial.jar` to
`/usr/$JAVA_BASE/jre/lib/ext`. Then make sure the user running the servant is in the groups `uucp`, `dialout` and `lock`, and that the package uucp is installed. Ensure the cross compilation environment for avr is installed. In Debian/Ubuntu these are the packages `gcc-avr`, `avr-libc` and `binutils-avr`.

Now check for the embedded C source in the subversion directory

```
svn+ssh://sagan/home/svnroot/Higgs/code/\
devices/arduino/arduino\_embedded
```

A copy of the Integrated Development Environment for the arduino is in

```
svn+ssh://sagan/home/svnroot/Higgs/code/devices/\
arduino/arduino-IDE
```

or you can get the latest version from the web at
`http://arduino.cc/en/Main/Software`. Start the IDE with `./arduino` and open the C source file `arduino_embedded.pde`. For the IDE to compile correctly, the name of the source file without the `.pde` extension must have the same name as the directory it is in. Select the correct Board and Serial Port [3] under the *Tools* menu, then Compile/Verify and Upload.

5.5 Power board

The Power Board is a custom printed circuit board designed specifically for the needs of the investigators at ASLab with which to test the algorithms in system auto-reprogramming with partial malfunction of a robot. The Power Board is in control of the power supply of up to nine devices in the robot. Each of these channels may be manually shut down by means of a switch or remotely/automatically using the ribbon cable connection to the Arduino board. There is a LED power indicator for each channel plus and a switch and LED indicator for all the board. Three of the channels, the ones nearer to the

[3]The MEGA Arduino board has an integrated USB to RS232 chip, so it will appear as a serial port `devttyUSBx` when connecting the USB cable.

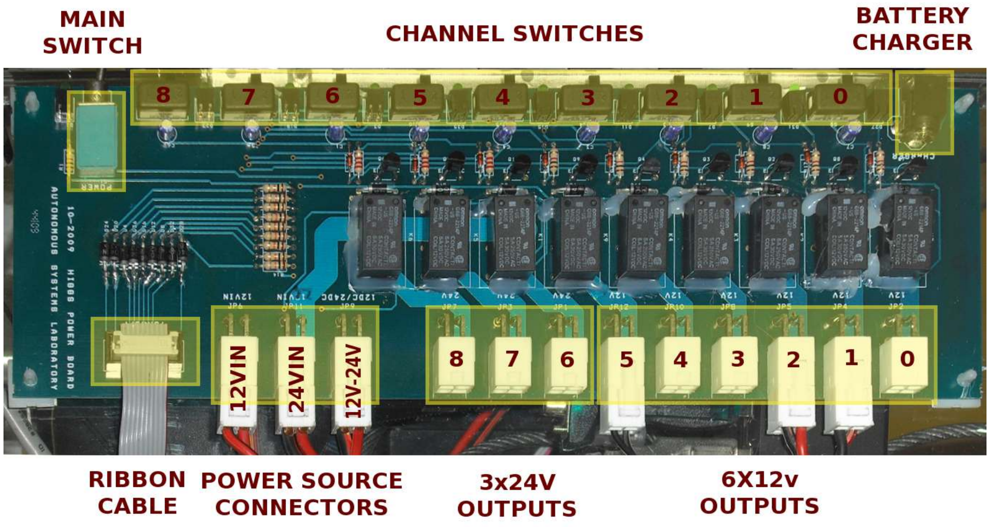


Figure 5.13: Photograph of the Power Board indicating each part.

general switch, can control 24V devices whilst the other 6 are for 12V devices. On the back side there are 12 connectors, 6x12V relay controlled outputs, 3x24V relay controlled outputs, 12V input, 24V input and a 12V output to be taken to the 12VDC to 24VDC converter that will be powered on whenever any 24V output is enabled. Additionally, there is a power jack connector for the battery charger[4]. Thus, the Power Board needs a 12V power source and if the 24V channels are used, it also needs a 12VDC to 24VDC converter.

5.5.1 Power board pinouts

Check figure 5.13 for a visual description of the connectors. The connector for the ribbon cable has the pinout indicated by table 5.3.

The connectors for the devices are Molex MiniFit, RS references 670-5717 (PCB male), 679-5776 (female), 172-9134 (terminals).

5.5.2 Electric interface

The ribbon cable connector has 10 pins. Starting from pin 0, these are ground, the six 12V channels and the three 24V channels. The channels are shut down writing a logical 0 (0V) to the corresponding pin, whilst a 1 (5V), a high impedance or a no connection will allow for the channel to be powered on. Both manual and remote switches must allow for the channel to be powered on for having that channel powered.

[4]Originally there was a lead battery pack for powering the devices on top of the robot base. This power jack connector is no longer neccessary as the robot base has its own power jack connector for charging the batteries.

Channel / Ribbon cable pin	Voltage	Device
0	—	Ground common
1	12V	Not used
2	12V	Arduino Extension Board (sensors)
3	12V	Servo
4	12V	Kinect
5	12V	GPS receiver
6	12V	Binocular Camera
7	24V	Laser
8	24V	Wrist
9	24V	Not used

Table 5.3: Distribution of devices in the channels.

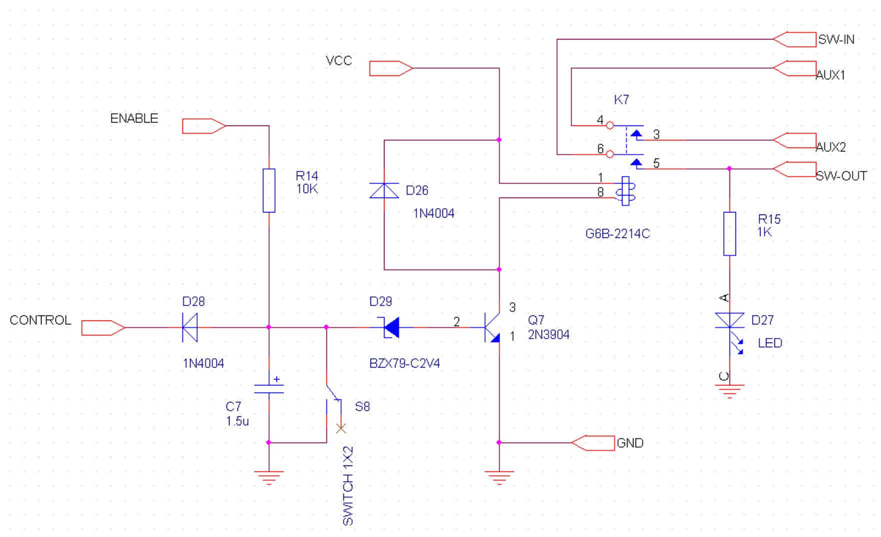


Figure 5.14: Schematic for each of the channels.

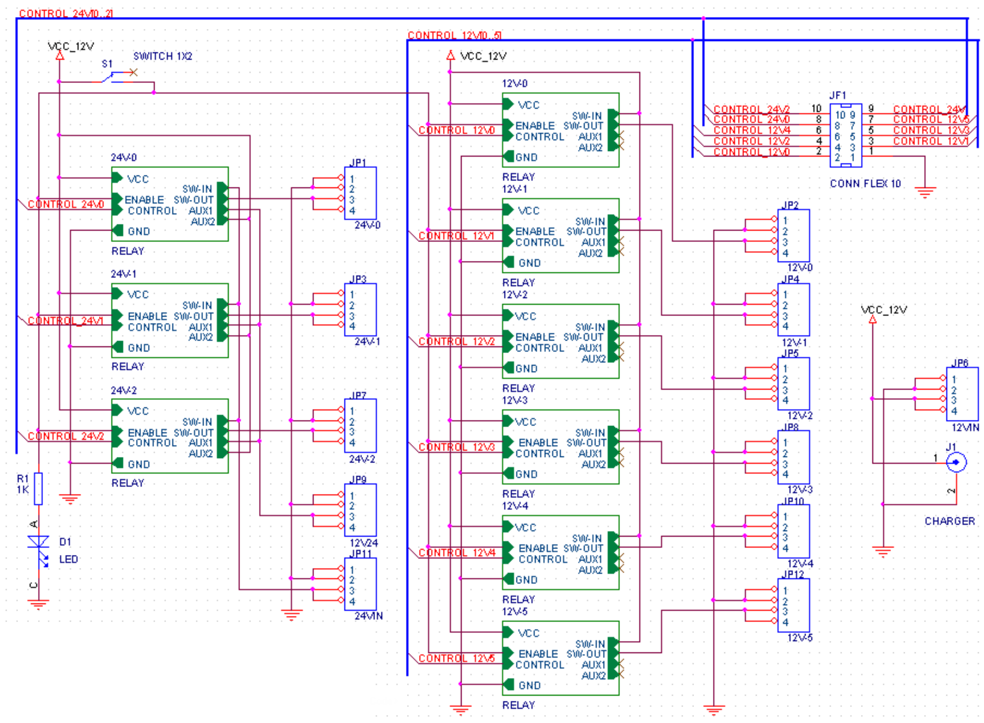


Figure 5.15: Schematic for the Power Board.

5.5.3 Power Board Bugs

During the design phase of the board there were some errors that passed the internal tests. As the board was to be manufactured once, it was not economically feasible to built it again and had to be repaired. These bugs should be revised before sending the design in case the power board should be manufactured again.

- The holes for the diodes 1N4004 are too small. Can be repaired by drilling bigger holes and solding the diodes on both sides.
- The power jack has the positive pin disconnected. Solved with a bit of tin covering both the connected pin hole and the positive pin hole.
- The silkscreen of the connector for the 24V input is wrong, it has a duplicated 12VIN instead of 24VIN. Solved with a marker-pen.
- The relay hole distribution has the two rows of pins too far apart. Moreover, the coils had positive and negative pins and was not well documented in the datasheet, so the coil pins are swapped. Solved by manually separating the pins of the relays and extending the coil pins and securing the relays with termoadhesive. A second better approach would have been to solder all components on the other side of the board.
- Sometimes the relays do not activate correctly and/or the external radio emitter for the DGPS interfere with them and turns them off. Even

though the calculations have been based on the components' specification, $R14$ and analogous have been reduced to $5K4\Omega$ for ensuring that enough current passes through the relay's coil.

5.6 Wrist

The wrist [5] is a two axis robotics kit module with pan and tilt movements manufactured by Schunk®. It was originally bought for use as the tilt mechanism for the laser, but as the center of gravity of the laser does not match the center of rotation of either axis of the wrist, it would be a very power hungry method for tilting the laser, given that the laser is heavy and the power comes from a portable battery system. Moreover, one of the axis from the wrist would be unused. It was finally decided to use the wrist for controlling the motion of the camera, even thought it could be achieved with a smaller controller with minor power requirements.

The manufacturer gives several interfaces for controlling the wrist: Profibus, CAN and serial. The serial RS-232 bus was chosen over the others because of the simplicity, the availability of drivers and the sufficient fulfillment of our requirements. It is connected to the on board computer via a custom made cable with an intermediate RS-2323 to USB converter. The wrist endpoint has industry standard serial closings.

5.6.1 Wrist servant

All source code for controlling the wrist is located under

```
\$(SVN\_ROOT)/Higgs/branches/CORBA/code/devices/wrist
```

The programs and utilities found there include:

- Low level library with direct access to serial port.
- Servant code for the CORBA object.
- Simple CORBA client to test the functionality and status.
- Graphical CORBA client for manually teleoperating the wrist with the mouse.
- Compressed file with the obsolete source code for the wrist, starting point but fully rewritten code for the current library.
- PDF file from the manufacturer describing the serial protocol to the wrist.

The sources are all under the `/src` directory and its documentation can be found inside the source files.

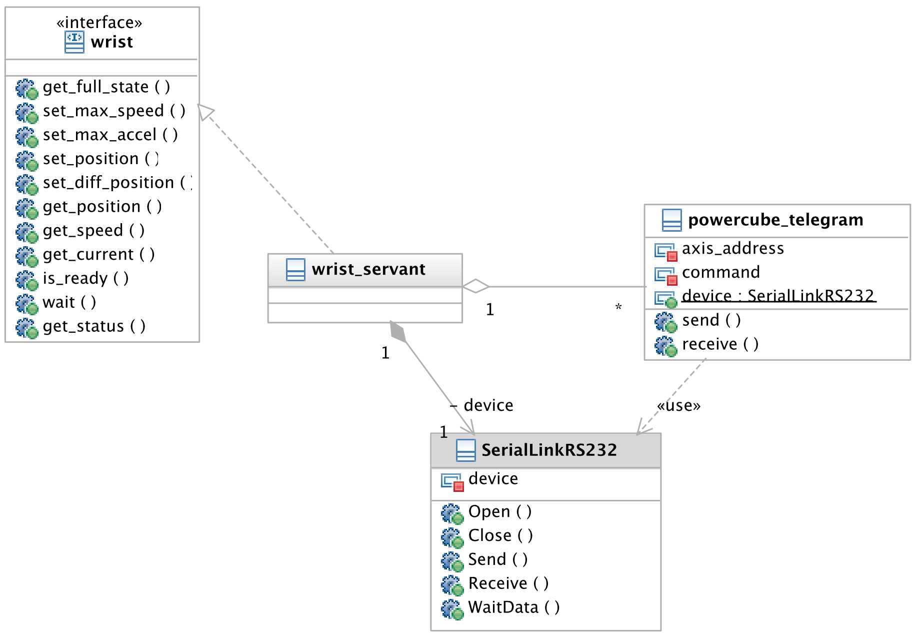


Figure 5.16: Class diagram for wrist module and CORBA interface

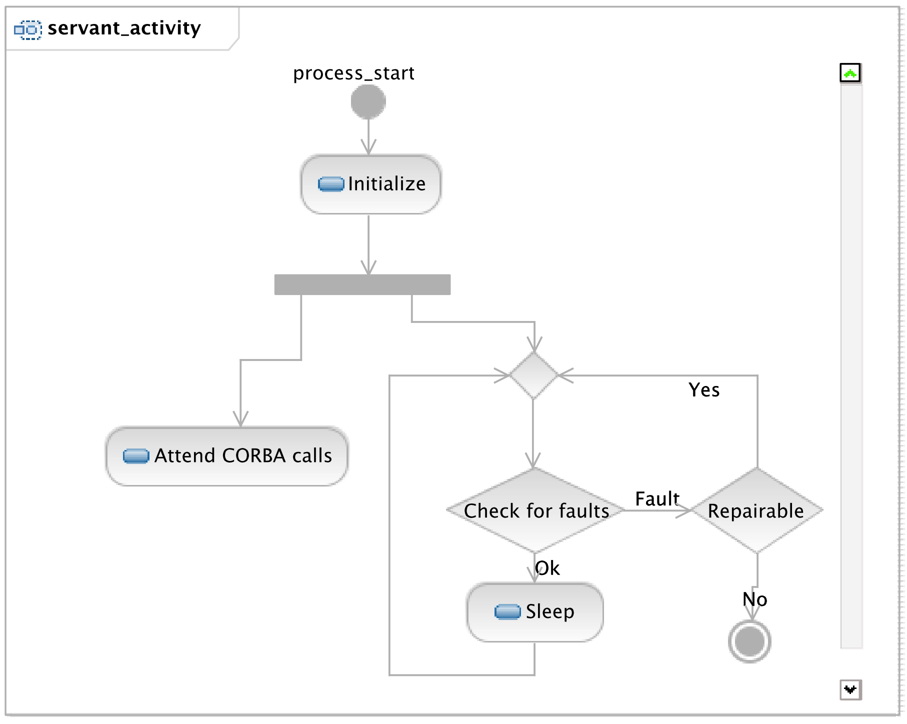


Figure 5.17: Activity diagram for wrist servant

When a CORBA client calls a method of this servant a new telegram is created with the parameters and format adequate to the call, passing the reference to the serial device already initialized. This telegram sends a message to the serial device and waits for the acknowledgement. This is valid for both directions of data flow. The telegram gets destroyed once the communication ends.

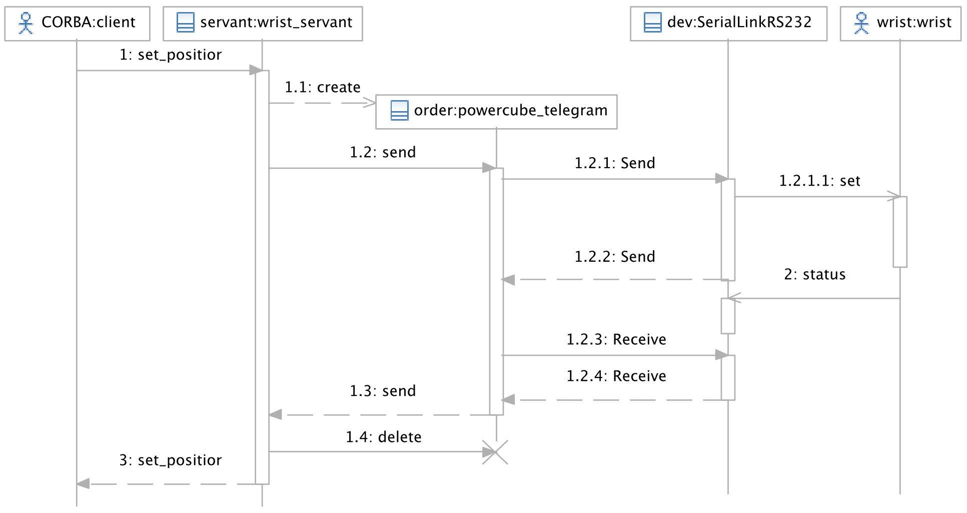


Figure 5.18: Sequence diagram for transmitting data to the wrist.

5.6.2 Error recovery

The wrist must be powered, as said in the official documentation, by a 24V power source, mobile or not. However, the fluctuations in voltage caused by battery charge and the instant consumption of the devices make it difficult to keep the voltage of the battery near this value. Because of this fact, the wrist controller has been designed for high error tolerance. A running thread in the servant polls periodically the wrist for error codes and takes the appropriate actions when a failure is detected, as explained in each considered fault. These are the possible faults that have been taken into consideration:

Battery voltage out of bounds

It has been detected empirically that the wrist will not function if the battery voltage is under 22V or if it is over 27V, so it will not work if the battery is either fully charged or next to empty. However, the control electronics are still available. The status command may be issued to detect this anomaly and a special status code will be sent by the wrist. The CORBA servant code is

[5]Note that even though the manufacturer calls this device a powercube, internally we call it wrist.

aware of this fault and will abort execution to allow to complete device reset, outputting a log and leaving the restart of the servant to the VAIO utility programs. This fault can be prevented by periodically reading the voltage sensor value from the I/O board, the Arduino CORBA servant.

Overcurrent

The internal current sensor will stop the axis when an overcurrent is detected or a low voltage condition is met if the 12V to 24V converter can not supply enough power. In this case, a reset and homing procedure is needed before continuing operating the unit. The maximum speeds and accelerations before the fault are saved and restored after the reset and homing procedure, but not the position.

Device not found

This error may arise if the device is powered off, if the serial port is not accessible or can not be found, or if the serial cable is not correctly connected. In this case the servant will die and get restarted automatically by the init scripts.

5.7 Laser

The data cable is a serial one prepared for RS-232 and RS-422 communications. There is a jumper on the end connector to select which of the protocols to use. With the jumper, RS-422. Without the jumper, RS-232. Note that the RS-422 has not been successfully tested, maybe because the jumper should be inside the laser connection black box instead of the serial cable endpoint.

5.7.1 Laser servant

The laser servant has been developed on top of the driver given by the people working in the mobile robotics lab. It has been modified to be able to resynchronize after a transmission error or a reboot of either the sensor or the servant. The original source code can be found in

```
$(SVN)/code/devices/laser/libreria_laser-paloma.zip
```

The documentation for the Sick Laser Sensor both hardwre and protocol definition can be found under the `doc` directory of the laser sources. Inside the `src` directory you will find the modified sources for the laser driver, the servant implementation and a test client, like in other modules.

5.8 Differential GPS

The GPS module is based on two OEMV-2-RT2 receptors with capacity for Satellite Based Augmentation System (SBAS) and Differential GPS (DGPS). Both receptors communicate through a radio based modem capable of transmitting in half-duplex mode through relatively long distances at 9600bps. The base radio has a transmitting power of 2W and the rover radio 0.5W, however the latter is only used for receiving data. Both radios must be set to work in the same channel, use the button with the channel label to change it. The electronics for each receptor have been encapsulated into plastic boxes with these connectors and indicators available:

- One double power LED indicator for checking 12V and 3.3V.
- One USB.
- Three RS-232 ports COM1 to COM3 with male DB9 connectors.
- One DB9 connector labelled CEXT.
- Power Jack connector, 12V nominal.
- Two TNC connectors.
- A reset button.

The connectors used on both base station and rover are the same. The antenna, either the small one on the rover or the bigger circular one on the base station is connected to the RFIN-labelled TNC connector. The other TNC connector is not used and its purpose is to have an external high precission oscillator for the GPS time measures. The power jack is used to power the unit. COM2 is connected to the radio emitter/receptor for sending/receiving the differential corrections from the base station to the rover. COM1 is connected to the VAIO laptop through a USB to RS232 adaptor so the GPS receiver can send the position information to the CORBA servant. On the base station, COM1 is only used when configuring the parameters of the receiver and saving them to its internal non-volatile memory.

5.8.1 Configuring the devices

These procedures are not to be used while on normal operation of the robot. They are only needed on first boot, then the configuration is saved permanently on the non-volatile memory of each device. There is a little booklet inside the yellow bag where the station base is stored with the commands and parameters used by the stations.

The `minicom` command line application is best used. You may have to configure it. Open `/etc/minirc.ttyS1` and insert these lines:

```
 # Machine-generated file - use "minicom -s" to change parameters.
pu port              /dev/ttyS1
pu baudrate          9600
pu bits              8
pu parity            N
pu stopbits          1
pu rtscts            No
```

Change the port as needed. Run

```
 minicom ttyS1
```

Now you should have access to a command line where you can send commands and check the status of the device.

Base station

Set up the base station with the RF input to the external antenna situated in the roof of the department of automatics. Get a serial cable and plug your personal computer to COM1. Start the `minicom` program to start a new session with the GPS device. These are commands were used to configure it:

```
FRESET
FIX POSITION 40.4397076 -3.6881482 744
INTERFACEMODE COM2 NONE CMR OFF
LOG COM2 CMROBS ONTIME 1
LOG COM2 CMRREF ONTIME 10
LOG COM2 CMRDESC ONTIME 10 1
SAVECONFIG
```

The meaning of the commands is as follows:

```
FRESET
```

Clears the non-volatile memory and sets the configuration to the default values. `RESET` does the same thing without clearing the non-volatile memory.

```
FIX POSITION 40.4397076 -3.6881482 744
```

This is the position measured for the base station after a period of several hours (Latitude, Longitude, Height). The command indicates the base station that it should calculate the GPS position for the corrections. All position reading after this command returns the fixed coordinates given.

```
INTERFACEMODE COM2 NONE CMR OFF
LOG COM2 CMROBS ONTIME 1
LOG COM2 CMRREF ONTIME 10
LOG COM2 CMRDESC ONTIME 10 1
```

Sets the COM2 port for sending the corrections using the message types `CMROBS`, `CMRREF` and `CMRDESC`. See the GPS manual for more information. `ONTIME 1` means to repeat the command every second. Use the `UNLOG log_name` and `UNLOGALL` commands to stop transmitting automatic logs. Omit `COM2` for sending the data your terminal.

```
SAVECONFIG
```

Stores the configuration to the non-volatile memory.

Rover station

Proceed as for the base station, except that the serial connection may be already available from the on board computer. In this case you may need to stop the GPS servant first to gain control of the serial device port. These commands were used to configure it:

```
FRESET
SBASCONTROL ENABLE EGNOS 0 ZEROTOTWO
INTERFACEMODE COM2 CMR NONE OFF
SAVECONFIG
```

The `SBASCONTROL` command line configures the GPS mobile station to use the EGNOS/SBAS satellites for better precision than GPS alone if the differential readings are not available.

5.8.2 Servant

The GPS module uses the convention for other module sources: installation and execution procedure, code placement inside the subversion repository and CORBA utilities such as in other modules are used.

The servant code uses non standard commands to receive the data from the rover station. The commands

```
LOG BESTPOS
LOG BESTVEL
```

are sent on startup and used to retrieve the information about satellites, position, standard deviation and speed. This data is fetched by an independent thread to the CORBA one and stores it into a shared variable for the CORBA thread to read it each time a client asks for the data. There is also a timeout by which if no data is received after a fixed amount of time an error is issued.

5.8.3 Radio signals and ETSII-UPM

The ETSII-UPM is surrounded by many governmental sites. It is possible that radio transmissions with the differential corrections fail because from interference from these sites. Other investigation groups have had the same problem. In this case the dGPS will not work. Official service has anyway recommended us to use this configuration in the rover if the problem persists:

```
UNDULATION USER 0.0
FIX NONE
COM COM2 9600 N 8 1 N OFF
LOG COM2 GPGGA ONTIME logperiod
INTERFACEMODE COM2 CMR NOVATEL OFF
RTKSOURCE type any (ANY)
```

5.9 Binocular camera

The binocular camera is attached to the top of the powercube, allowing for pan and tilt movements.

The image data is transmitted in raw yuv format between the CORBA servant and the clients.

When restarting the servants, you should kill the process and call `dc1394_reset_bus`, or else the servant will not succeed on starting again.

5.9.1 Compiling and running the CORBA module

For installing the CORBA module on a fresh fedora linux system, you must install these libraries which can be found in the repositories provided by the package `atrpms`: `glut glui libavcodec` and related. Inside the directory tree of the sources there are three scripts that help run the servant and test clients. Compile and run the server with

```
\$ ./1-server.sh
```

The test client with

```
\$ ./2-client.sh
```

or, for the old client,

```
\$ ./3-client.sh
```

The client runs best when configured to run with Xlib instead of OpenGL. The user that runs the servant, higgs when running unattended, needs to be in the video group.

The source structure has not been adapted to the new tree configuration nor cmake. `idl` Contains the idl interfaces. `generated` is where the skeleton and stub source files are generated to. `obj` the binary files are compiled to this directory. `text` contains code to test the camera without using CORBA.

5.10 BatteryModel and CurrentMonitor

Note: Porting to the newer SVN repository is pending.

These two CORBA servants have been implemented into one JAVA binary and made an unique module inside the VAIO on board computer. The BatteryModel is a component that calculates the parameters related to the charge in a battery and estimates the remaining life based on the power requirements. The CurrentMonitor is not used in any other module but has been an utility component for developing the BatteryModel. It periodically polls the current of the (currently instrumentation) batteries and reports the mean consumption since it was required to start monitoring.

Neither component reads or uses the peripherals on the computer. BatteryModel is exclusively a CORBA servant and the clients must pass the data to it in order to compute the estimations for the battery life, whilst CurrentMonitor is, besides a servant, a client to the Arduino CORBA servant for reading the current values of the batteries.

Installation of the module and automatic setup is achieved as in the other modules.

Chapter 6

ASys Approach to Higgs

The *ASys Engineering Methodology* developed in ICEA is based on the OASys ontology and has been applied to the Robot Control Testbed for its conceptual analysis. The following sections provide a description on how the different tasks were accomplished, with examples of the different workproducts obtained. The terms in italic that appear in the phases, tasks, subtasks and models descriptions are those defined in the Packages of OASys.

6.1 RCT ASys Views Phase

The purpose of this *Phase* is to identify the different views of interest for the RCT considering the Perspective Package elements, specialised through the ASysPerspective Package concepts.

For example the *RequirementViewpoint* especilises the concept of *Viewpoint* to address the analysis of the RCT from a *ASysRequirement Concern*, to establish the different requirements to be defined in the system. Considering this viewpoint, a RCTRequirementViewpointModel can be instantiated (Fig. 6.1). The different concepts, relationships and attributes where chosen from the Perspective and the ASysPerspective Package. The original concepts have been ontologically instantiated into RCT related ones. For example, the concept RCTRequirement is an *ASysRequirement* concern, which in turn is a*Concern* from the Perspective Package. In a similar way, RCTPerformanceViewpoint is a *Viewpoint*, as defined in the Perspective Package.

The *StructuralViewpoint* pays attention to the au- tonomous system structure, as considering the *ASysStructure* concern. A RCTStructuralViewpointModel can be obtained (Fig. 6.1) by instantiating the different concepts, relationships and attributes in the Perspective and the ASysPerspective Packages. In this model, the RCTStructuralView is a *View* that will consist of a RCTStructuralModel, which is the *EngineeringModel* to be obtained as output of the StructuralAnalysis task.

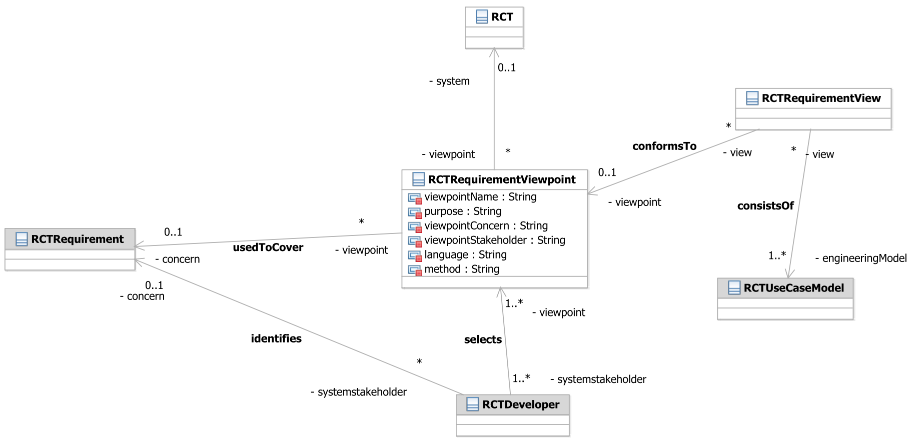


Figure 6.1: RCT RequirementViewpoint Model

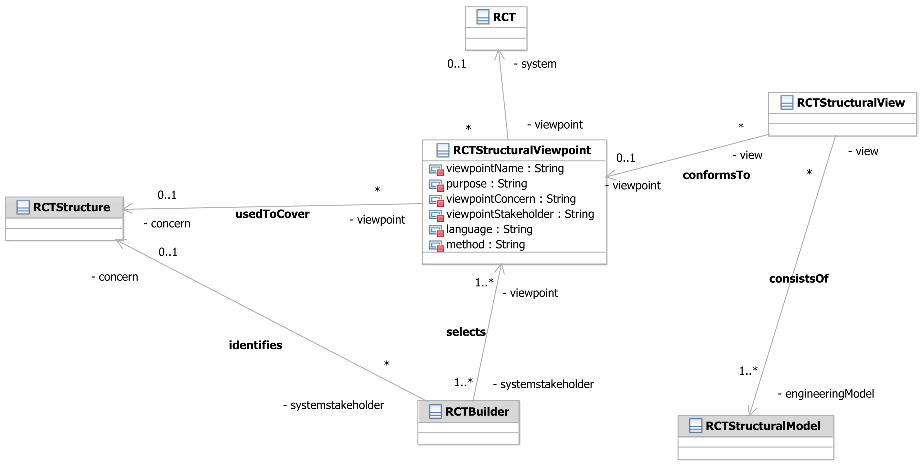


Figure 6.2: RCT StructuralViewpoint Model

Additionally, the *BehaviouralViewpoint* describes the autonomous system's behaviours, i.e., how the system operates under certain conditions. The RCT-BehaviouralViewpointModel obtained (Fig. 6.3) instantiates the concepts and relationships in the Perspective and the ASysPerspective Packages. The different roles played for the concepts in the model are shown in each one of the relationships, which are given the same name as in the original package.

In turn, the *FunctionalViewpoint* focus on the analysis of an autonomous system's functions, as desired or expected behaviours. The RCTStructuralViewpointModel (Fig. 6.4) instantiates the Perspective and the ASysPerspective Packages ontological elements, focusing on the functional aspects to consider for the RCT. In this model, the RCTFunctionalView is a *View* formalises as

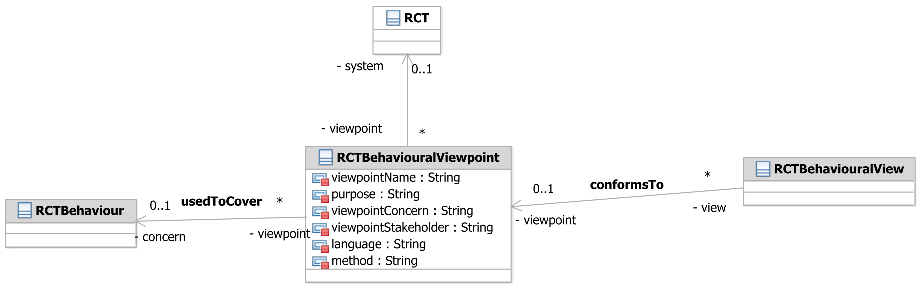


Figure 6.3: RCT BehaviouralViewpoint Model

a RCTFunctionalModel, which is the *EngineeringModel* consisting of different function related models, obtained as outputs of the FunctionalModelling subtask.

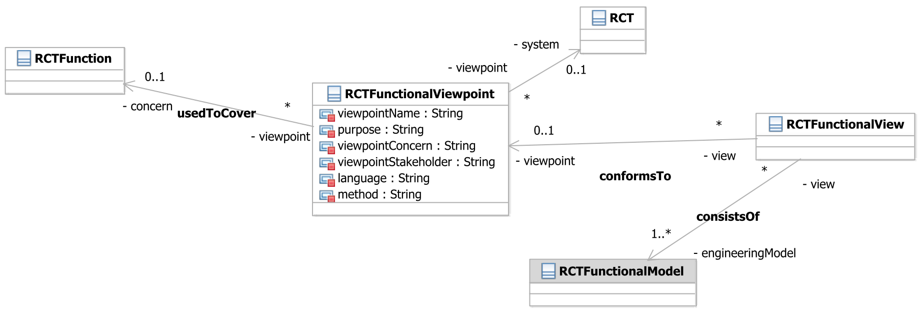


Figure 6.4: RCT FunctionalViewpoint Model

During the RCT development, it will be necessary to address the *Performance-Viewpoint*, to evaluate the performance requirements and benchmarking of the RCT once it is finally implemented. The PerformanceViewpoint addresses the performance aspects of the RCT. The Perspective and the ASysPerspective Package ontological elements were used to build up a PerformanceViewpointModel for the RCT (Fig. 6.5).

6.2 RCT ASys Requirement Phase

The purpose of this phase is to identify and elicit stakeholders' requirements for the RCT considering the RequirementViewpoint. The requirements are to be specified during this phase, by using traditional requirements engineering techniques.

- System UseCase Task

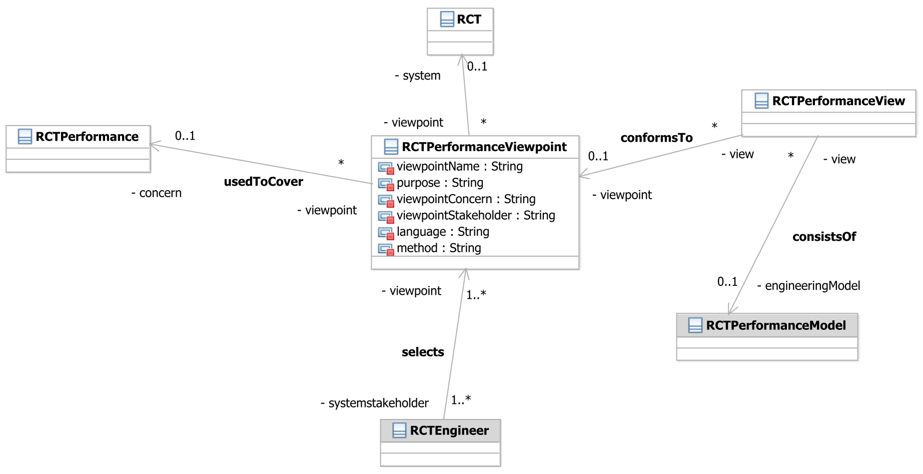


Figure 6.5: RCT PerformanceViewpoint Model

The first *Task* during the *ASys Requirement Phase* is to analyse the *System UseCase*s, by instantiating the Requirement Package' ontological elements in the SystemEngineering Subontology. This *Task* consists of the *Subtask*s of *UseCaseModelling* and *UseCaseDetailing*, as defined in the ASys Engineering Subontology.

An *UseCase* in OASys has been defined as a mean to capture a requirement of a system, as defined in the Perspective Package.To define the UseCase, the *Subject* as system under consideration, and the different *UseCaseActor*s as objects that interact with the system are also identified, among other aspects. As a result, a *SystemUseCaseModel* is obtained, detailing the previous identified elements. The UML classes in the model are instantiation of the original OASys concepts, this fact being specified by the UML roles names in the shown associations.

When the *Subject* under study is the Robot Control Testbed, an RCT *UseCaseModel* shows the RCT's requirements by means of use cases. A system's requirements can be of different types (physical, functional, performance, interface, design) as defined in the Requirement Package. An initial requirement analysis made by the RCT developers, identified the *FunctionalRequirement*s for the RCT, where this concept has been defined as a "requirement that specifies an operation or behaviour that a system must perform" in OASys. Primary *FunctionalRequirement*s for the RCT are to navigate, as well as to survive.

The navigation requirement is captured by means of the *UseCase* Navigation, which *include*s the secondary *FunctionalRequirement*s of being able to explore the environment, identify elements in the environment, and to avoid obstacles. These requirements are captured in the *Usecase*s of EnvironmentExploration, Identification and ObstacleAvoidance respectively (Fig. 6.6). In turn, the *FunctionalRequirement* of surviving is captured in the Survival *UseCase*, which *include*s the SubsystemFailure and Recharge *UseCase*s (Fig. 6.7).

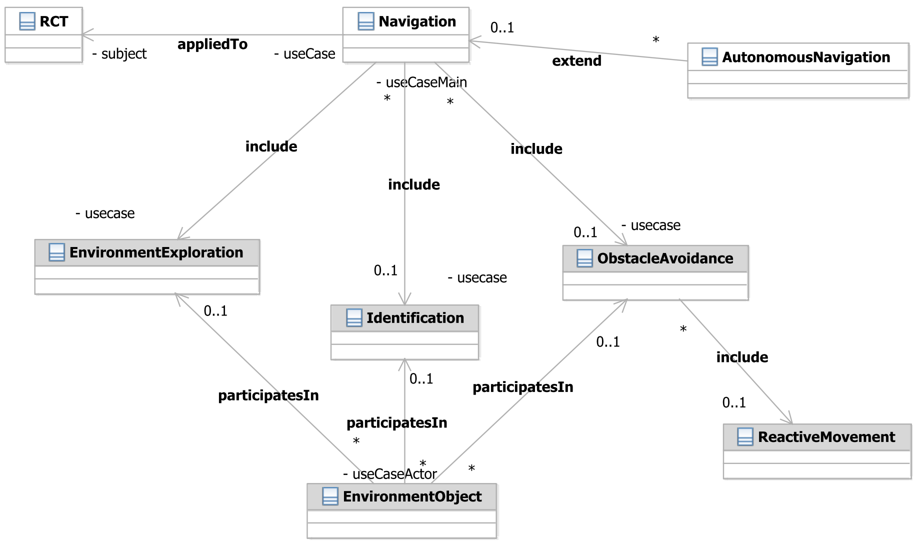


Figure 6.6: RCT Navigation UseCaseModel

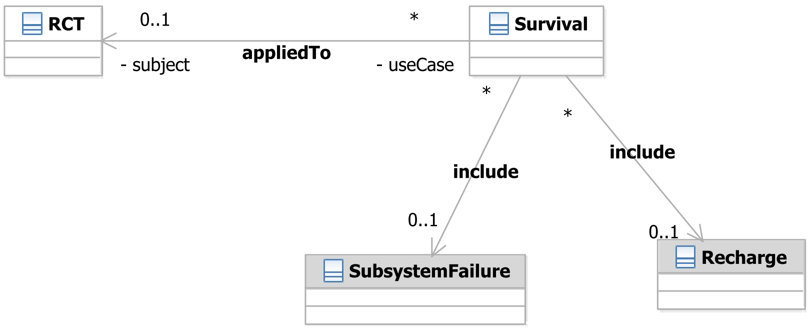


Figure 6.7: RCT Survival UseCaseModel

It might be interesting to detail a particular *UseCase* by paying attention to the *Subsystems* in the *System*, to detail further *Requirement*s. *Subsystems* identified in the RCT are the BasePlatform, the OnboardSystem, and the SupportingSystem. Focusing, for example, on the Navigation *UseCase* previously defined, it is possible to identify additional *Requirement*s for the *Subsystem*s, in the form of a RCT Subsystem *UseCaseModel* (Fig. 6.8).

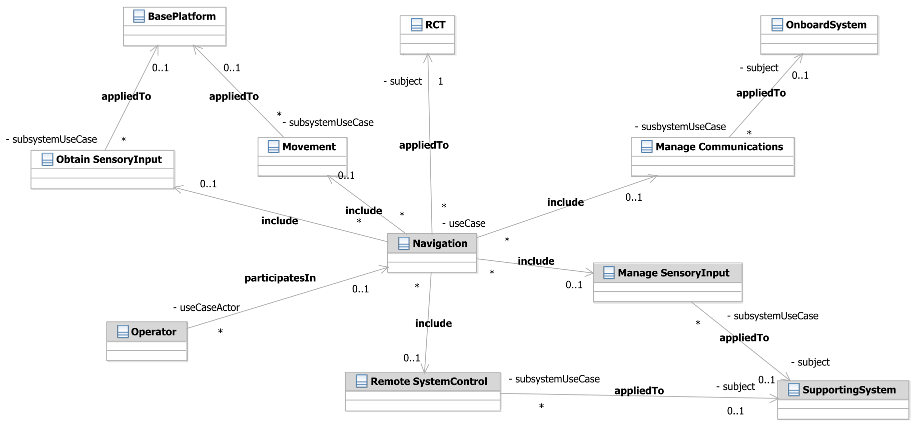


Figure 6.8: RCT Navigation UseCaseModel detailed for Subsystems

The *UseCase Modelling Subtask* can also obtain a *Subsystem UseCaseModel*, which gathers the *Requirement* for a particular *Subsystem*. In the case of the RCT, this subsystem usecasemodel was obtained for the BasePlatform (Fig. 6.9 showing the instantiation of the original ontological elements) to detail the *FunctionalRequirement* of movement; for the OnboardSystem to specify the *FunctionalRequirement* of ManageCommunications (Fig. 6.10); and for the SupportingSystem to clarify the Manage SensoryInput requirement (Fig. 6.11).

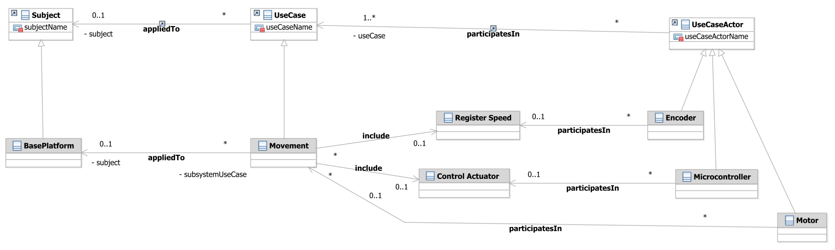


Figure 6.9: BasePlatform Subsystem UseCaseModel

The next *Subtask* in the *ASysRequirement Phase* is to detail each *UseCaseModel* obtained during a *UseCase Detailing Subtask*. This detail can be achieved by filling a pre–defined ASLab UseCase Pattern, with the relevant information in each field as required. This *Subtask* was performed by the RCT developers, having as results different textual tables for the previous *UseCase*s. For example, the detail for the Navigation requirement is shown in Fig. 6.12, and the Survival one in Fig. 6.13.

- Requirement Characterisation

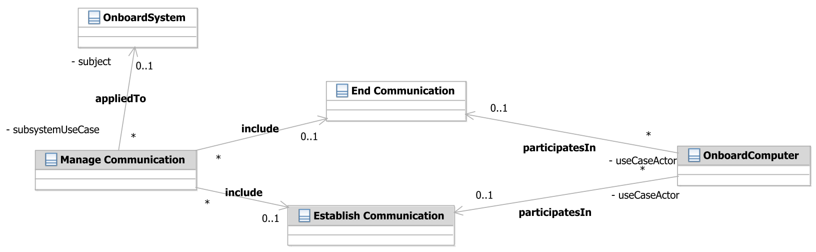


Figure 6.10: OnboardSystem Subsystem UseCaseModel

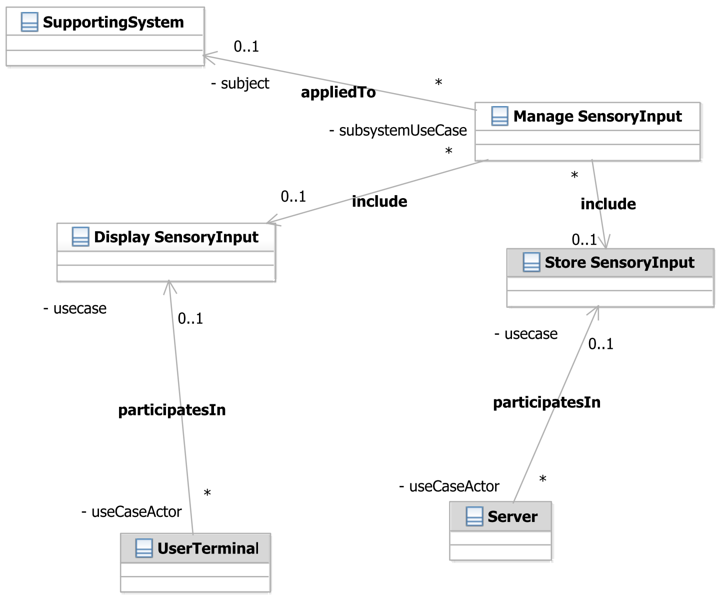
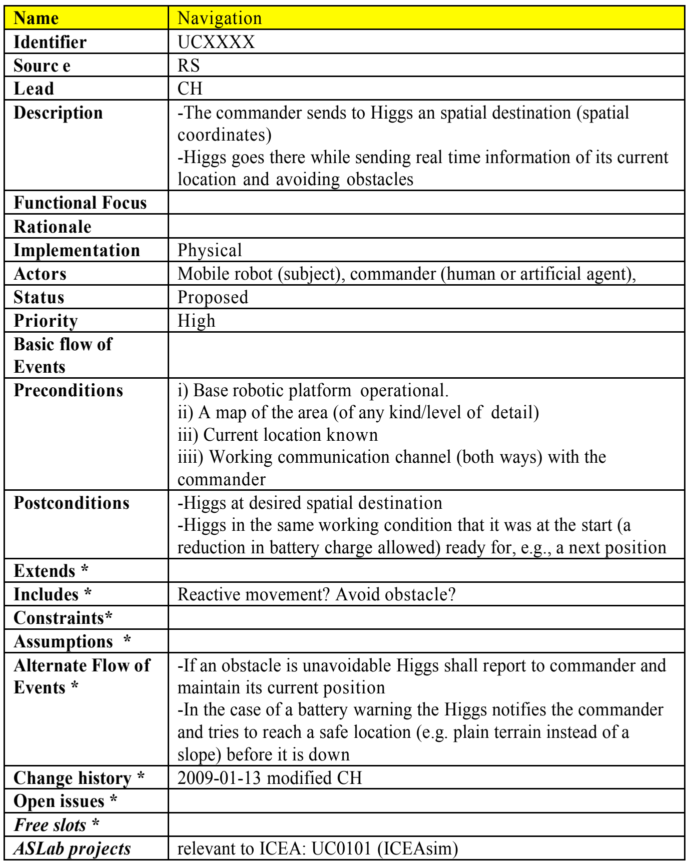
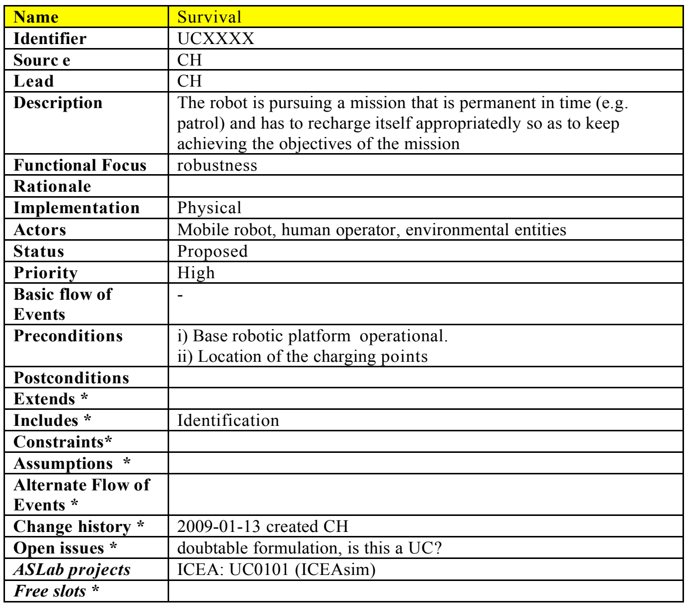


Figure 6.11: SupportingSystem Subsystem UseCaseModel



Name	Navigation
Identifier	UCXXXX
Sourc e	RS
Lead	CH
Description	-The commander sends to Higgs an spatial destination (spatial coordinates) -Higgs goes there while sending real time information of its current location and avoiding obstacles
Functional Focus	
Rationale	
Implementation	Physical
Actors	Mobile robot (subject), commander (human or artificial agent),
Status	Proposed
Priority	High
Basic flow of Events	
Preconditions	i) Base robotic platform operational. ii) A map of the area (of any kind/level of detail) iii) Current location known iiii) Working communication channel (both ways) with the commander
Postconditions	-Higgs at desired spatial destination -Higgs in the same working condition that it was at the start (a reduction in battery charge allowed) ready for, e.g., a next position
Extends *	
Includes *	Reactive movement? Avoid obstacle?
Constraints*	
Assumptions *	
Alternate Flow of Events *	-If an obstacle is unavoidable Higgs shall report to commander and maintain its current position -In the case of a battery warning the Higgs notifies the commander and tries to reach a safe location (e.g. plain terrain instead of a slope) before it is down
Change history *	2009-01-13 modified CH
Open issues *	
Free slots *	
ASLab projects	relevant to ICEA: UC0101 (ICEAsim)

Figure 6.12: UseCase Detailing Subtask for the Navigation requirement



Name	Survival
Identifier	UCXXXX
Sourc e	CH
Lead	CH
Description	The robot is pursuing a mission that is permanent in time (e.g. patrol) and has to recharge itself appropriatedly so as to keep achieving the objectives of the mission
Functional Focus	robustness
Rationale	
Implementation	Physical
Actors	Mobile robot, human operator, environmental entities
Status	Proposed
Priority	High
Basic flow of Events	-
Preconditions	i) Base robotic platform operational. ii) Location of the charging points
Postconditions	
Extends *	
Includes *	Identification
Constraints*	
Assumptions *	
Alternate Flow of Events *	
Change history *	2009-01-13 created CH
Open issues *	doubtable formulation, is this a UC?
ASLab projects	ICEA: UC0101 (ICEAsim)
Free slots *	

Figure 6.13: UseCase Detailing Subtask for the Survival requirement

An additional *Task* in this *Phase* is to characterise the autonomous system's requirements, to obtain a *RequirementCharacterisation*, as defined in the ASys Engineering Process Package of the ASys Engineering Subontology. The taxonomies of concepts considered in the ASys Requirement Package can be described specifying a condition, a requirement criterion and a possible threshold. As a result, a *Process Characterisation* and a *System Characterisation* in the form of a textual description.

6.3 RCT ASys Analysis Phase

As part of the ASys Analysis Phase consists of differents *Tasks*, such as *StructuralAnalysis*, *BehaviouralAnalysis*, and *FunctionalAnalysis*. The differents tasks have been carried out for the Robot Control Testbed, as described in following sections.

Structural Analysis

The *StructuralAnalysis Task* pays attention to the system under analysis from a *StructuralViewpoint*, i.e., the analysis considering the system's structure in terms of subsystems and elements. As outcome of the task, a *StructuralModel* for the RCT is obtained, consisting of *StructureModel*s and*Topology Model*s.

- System Modelling

  The *SystemModelling Subtask* carried out for the RCT has obtained the *StructuralModel*, consisting of the *StructureModel*s and the *TopologyModel*s. For the RCT *StructureModel*, the different subsystems part of it were determined according to the structural definition provided. Hence, three different kind of subsystem were identified: the platform, the onboard system attached to it, and the supporting system which are not physically part of the RCT but take part into its operation. The platform was further analysed to identify the subsystems or elements part of it. The different onboard systems were identified. Finally, the supporting systems were specified. The overall analysis result was formalised as a *StructureModel* for the RCT (Fig. 6.14).

  The *TopologyModel* allows representing the topological connections between a system's parts, by refining the Topology Package concepts. For the Robot Control Testbed, it was considered important to differentiate the topology from an informational and a physical

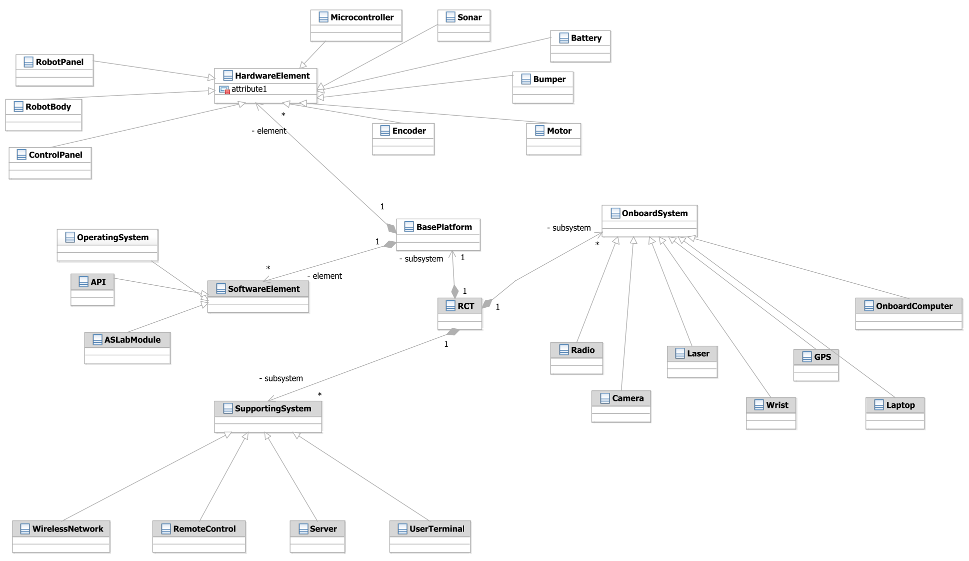


Figure 6.14: RCT Structure Model

aspects. The first one refers to the communication connections between parts, such as WiFi, Ethernet, etc. The latter, the physical connections among parts in a traditional way. Hence, two different topology models have been obtained for the RCT system: the RCT Informational Topology Model (Fig. 6.15), and the RCT Physical Topology Model (Fig. 6.16).

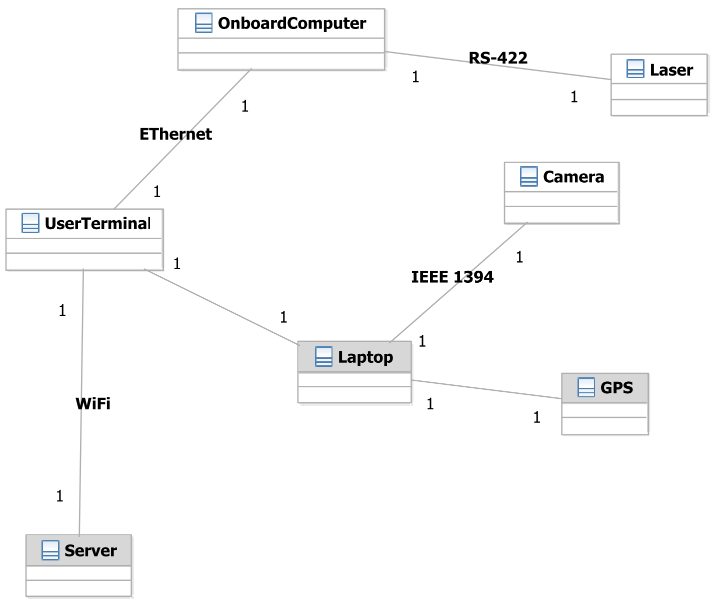


Figure 6.15: RCT Topology Model: informational connections

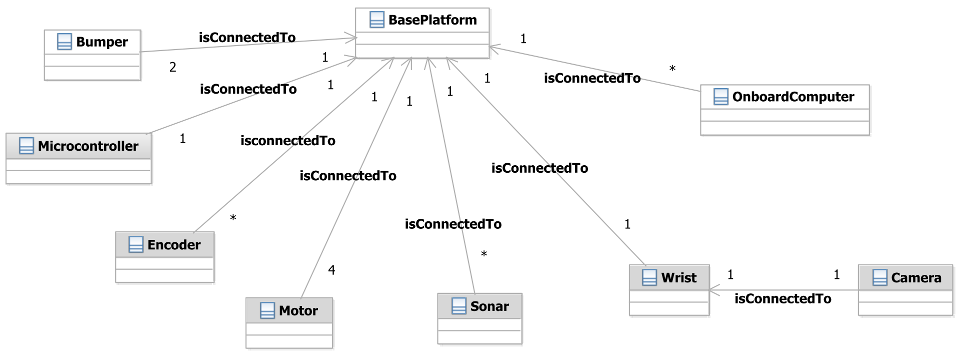


Figure 6.16: RCT Topology Model: physical connections

Each one of the subsystems and elements identified in the *StructureModel*, can be further detailed during a design phase to represent their characteristics and roles, by ontologically instantiating the concepts in the PhysicalDevice Package into a *DeviceModel*.

Chapter 7

Mission Specification

Chapter 8

Higgs Webots Model

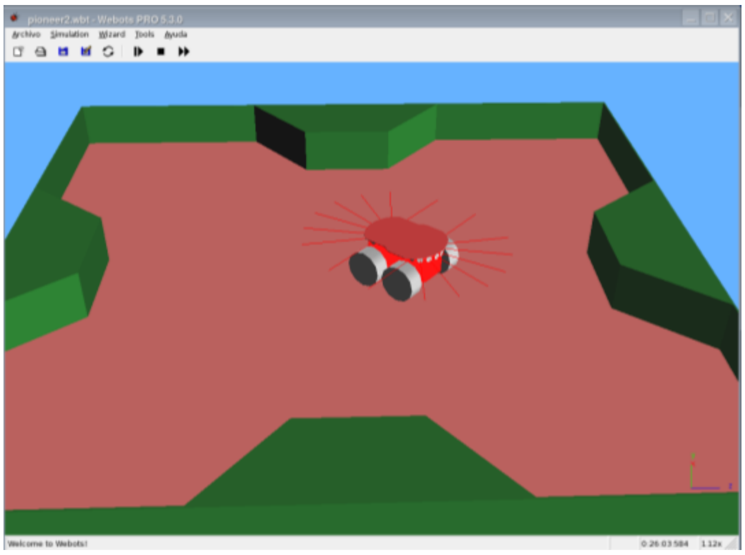

pioneer2.wbt - Webots PRO 5.3.0
Archivo Simulation Wizard Tools Ayuda
Welcome to Webots!

Figure 8.1: The Higgs virtualisation in Webots.

Chapter 9

Model-based Synthesis

This chapter will contains the description of the software synthesis process from models.

Chapter 10

Higgs Self-Awareness

This chapter will contains the description of the mechansisms for self-awareness in Higgs.

Appendix A

CORBA Interface Definitions

Appendix B

XML Self-model

The self-management system uses an XML model as the central repository of information on reflective knowledge. This is used by the self manager in the decision making concerning the functional operation of the robot.

What follows is the XML model concerning bandwidth-based reconfiguration of the robot perception system.

```
<?xml version="1.0" encoding="UTF-8"?>
<system xmlns="http://www.rcc.com/system"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.rcc.com/system file:/C:/Users/raduadmin

    <name>BandwithControl</name>

    <resource>
        <ID>robot</ID>

<!-- state parameter section -->

        <property>
            <ID>link_quality</ID>
            <propertyDataType>
                <xs:element name="link_quality" type="robotLink_quality" /
                <xs:simpleType name="robotLink_quality">
                    <xs:restriction base="xs:int"/>
                </xs:simpleType>
            </propertyDataType>
            <mutability>constant</mutability>
            <modifiability>read-only</modifiability>
            <subscribeability>false</subscribeability>
            <primaryKey>false</primaryKey>
        </property>
        <property>
            <ID>bit_rate</ID>
            <propertyDataType>
                <xs:element name="bit_rate" type="robotBit_rate" />
                <xs:simpleType name="robotBit_rate">
```

```
                        <xs:restriction base="xs:int"/>
                    </xs:simpleType>
                </propertyDataType>
                <mutability>constant</mutability>
                <modifiability>read-only</modifiability>
                <subscribeability>false</subscribeability>
                <primaryKey>false</primaryKey>
            </property>
            <property>
                <ID>missed_beacons</ID>
                <propertyDataType>
                    <xs:element name="missed_beacons" type="robotMissed_beaco
                    <xs:simpleType name="robotMissed_beacons">
                        <xs:restriction base="xs:long"/>
                    </xs:simpleType>
                </propertyDataType>
                <mutability>mutable</mutability>
                <modifiability>read-only</modifiability>
                <subscribeability>false</subscribeability>
                <primaryKey>false</primaryKey>
            </property>
            <property>
                <ID>video_fps</ID>
                <propertyDataType>
                    <xs:element name="video_fps" type="robotVideo_fps" />
                    <xs:simpleType name="robotVideo_fps">
                        <xs:restriction base="xs:double"/>
                    </xs:simpleType>
                </propertyDataType>
                <mutability>constant</mutability>
                <modifiability>read-only</modifiability>
                <subscribeability>false</subscribeability>
                <primaryKey>false</primaryKey>
            </property>

        <!-- derived parameter section -->

        <property>
        <ID>bwidth_used</ID>
        <propertyDataType>
        <xs:element name="bwidth_used" type="robotBwidth_used" />
        <xs:simpleType name="robotBwidth_used">
        <xs:restriction base="xs:int"/>
        </xs:simpleType>
        </propertyDataType>
        <mutability>constant</mutability>
        <modifiability>read-only</modifiability>
        <subscribeability>false</subscribeability>
        <primaryKey>false</primaryKey>
        </property>


        <!-- configuration parameter section -->
```

```
            <property>
                        <ID>video_rgb</ID>
                        <propertyDataType>
                            <xs:element name="video_rgb" type="robotVideo_rgb" />
                            <xs:simpleType name="robotVideo_rgb">
                                <xs:restriction base="xs:int"/>
                            </xs:simpleType>
                        </propertyDataType>
                        <mutability>constant</mutability>
                        <modifiability>read-write</modifiability>
                        <subscribeability>false</subscribeability>
                        <primaryKey>false</primaryKey>
                    </property>
                    <property>
                        <ID>video_width</ID>
                        <propertyDataType>
                            <xs:element name="video_width" type="robotVideo_width" />
                            <xs:simpleType name="robotVideo_width">
                                <xs:restriction base="xs:int"/>
                            </xs:simpleType>
                        </propertyDataType>
                        <mutability>constant</mutability>
                        <modifiability>read-write</modifiability>
                        <subscribeability>false</subscribeability>
                        <primaryKey>false</primaryKey>
                    </property>
                    <property>
                        <ID>video_trans</ID>
                        <propertyDataType>
                            <xs:element name="video_trans" type="robotVideo_trans" />
                            <xs:simpleType name="robotVideo_trans">
                                <xs:restriction base="xs:int"/>
                            </xs:simpleType>
                        </propertyDataType>
                        <mutability>constant</mutability>
                        <modifiability>read-write</modifiability>
                        <subscribeability>false</subscribeability>
                        <primaryKey>false</primaryKey>
                    </property>
                    <property>
                        <ID>laser_trans</ID>
                        <propertyDataType>
                            <xs:element name="laser_trans" type="robotLaser_trans" />
                            <xs:simpleType name="robotLaser_trans">
                                <xs:restriction base="xs:int"/>
                            </xs:simpleType>
                        </propertyDataType>
                        <mutability>constant</mutability>
                        <modifiability>read-write</modifiability>
                        <subscribeability>false</subscribeability>
                        <primaryKey>false</primaryKey>
                    </property>
                    <property>
                        <ID>sonars_trans</ID>
```

```
                <propertyDataType>
                    <xs:element name="sonars_trans" type="robotSonars_trans"
                    <xs:simpleType name="robotSonars_trans">
                        <xs:restriction base="xs:int"/>
                    </xs:simpleType>
                </propertyDataType>
                <mutability>constant</mutability>
                <modifiability>read-write</modifiability>
                <subscribeability>false</subscribeability>
                <primaryKey>false</primaryKey>
            </property>

<!-- operational model section -->

<property>
<ID>operationalModel</ID>
<propertyDataType>
<xs:element name="operationalModel" type="robotOperationalModel"/>
<xs:complexType name="robotOperationalModel">
<xs:sequence>
<xs:element name="modelElement" type="robotModelElement" maxOccurs="unbou
</xs:sequence>
</xs:complexType>
<xs:complexType name="robotModelElement">
<xs:sequence>
<xs:element name="bwidth_used" type="robotBwidth_used" />
<xs:element name="video_rgb" type="robotVideo_rgb" />
<xs:element name="video_width" type="robotVideo_width" />
<xs:element name="video_trans" type="robotVideo_trans" />
<xs:element name="laser_trans" type="robotLaser_trans" />
<xs:element name="sonars_trans" type="robotSonars_trans" />
<xs:element name="video_fps" type="robotVideo_fps" />
</xs:sequence>
</xs:complexType>
</propertyDataType>
<mutability>constant</mutability>
<modifiability>read-only</modifiability>
<subscribeability>false</subscribeability>
<primaryKey>false</primaryKey>
</property>

</resource>
</system>
```

Bibliography

Title: Higgs Manual
Subtitle: A Platform for Autonomy Research
Author: Carlos Hernández, Adolfo Hernando, Ricardo Sanz and Francisco Arjonilla

Date: December 5, 2011
Reference: R-2010-008 v 0.1 Draft

URL: http://www.aslab.org/documents/controlled/ASLAB-R-2010-008.pdf

Autonomous Systems Laboratory

UNIVERSIDAD POLITÉCNICA DE MADRID
C/JOSÉ GUTIÉRREZ ABASCAL, 2
MADRID 28006 (SPAIN)